VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Webové rozhraní pro monitoring call centra na bázi Asterisk PBX

Web Monitoring Interface for Call Center based on Asterisk PBX

2019 Adam Molin

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Zadání bakalářské práce

Student: **Adam Molin**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R059 Mobilní technologie

Téma: Webové rozhraní pro monitoring call centra na bázi Asterisk PBX
Web Monitoring Inteface for Call Center based on Asterisk PBX

Jazyk vypracování: čeština

Zásady pro vypracování:

Práce je zaměřena na návrh a implementaci odlehčeného webového monitorovacího systému, který by mohl být využit v menších call centrech pro zobrazení dostupných linek a agentů, aktuálně probíhajících hovorů a dalších informací využitelných při správě call centra menší velikosti.

Cíle práce shrnují tyto body:
1. Student si nastuduje a zhodnotí architekturu a implementaci SIP serveru Asterisk, zejména pak ty jeho funkionality, které lze využít v případě call center.
2. Student definuje a popíše informace, které je třeba vyměňovat mezi Asterisk PBX a navrhovaným systémem.
3. S využitím standardizovaných rozhraní Asterisk PBX student navrhne a zrealizuje monitorovací systém call centra.
4. Student zhodnotí navržený systém z pohledu jeho využitelnosti, interoperability s jinými řešeními a nasaditelnosti v různých prostředích.

Seznam doporučené odborné literatury:

[1] Bryant, R. Madsen, L. Asterisk: The Definitive Guide. ISBN 1449332420.
[2] Hartpence, B. Packet Guide to Voice over IP: A system administrator's guide to VoIP technologies. ISBN 1449339670.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Rozhon, Ph.D.**

Datum zadání: 01.09.2017
Datum odevzdání: 30.04.2019

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky 1D16 OSTRAVA

prof. Ing. Miroslav Vozňák, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2019 .....

Rád bych na tomto místě poděkoval Ing. Janu Rozhonovi, Ph.D., za odbornou pomoc a konzultaci při tvorbě této bakalářské práce.

Abstrakt

Cílem této bakalářské je práce vytvoření webového rozhraní pro správu menších call center. Hlavní součástí práce je seznámení se s technologií VoIP a jejími protokoly, systémem Asterisk, který implementuje telefonní ústřednu (PBX) na běžném počítači, propojení s databázovým systémem MySQL a následná implementace webového serveru, který nám poskytuje webové rozhraní na základě logů CEL poskytnutých systémem Asteriskem. Výstupem práce je webová aplikace pro monitorování dostupných linek, probíhajících hovorů, front a dostupných agentů.

Klíčová slova: Asterisk, MySQL, CEL, webová aplikace, monitorování, call centrum

Abstract

Subject of this bachelor thesis is designing web application for monitoring and managing small call centers. Main points of the thesis are introduction to VoIP technology and it's protocols, Asterisk system, which implements private branch exchange (PBX) on regular computer, connection with MySQL database system and implementation of web server, which provides us with web interface based on CEL logs provided by Asterisk PBX. Result of the thesis is web application for monitoring available lines, ongoing calls, queues and available agents.

Key Words: Asterisk, CEL, MySQL, web application, monitoring, call center

Obsah

Seznam použitých zkratek a symbolů

AMA	–	Automated Message Accounting
ANI	–	Automatic Number Identification
API	–	Application Programming Interface
B2BUA	–	Back-to-Back User Agent
CEL	–	Channel Event Logging
CID	–	CallerID
CLI	–	Command-line interface
CSS	–	Cascade Style Sheets
CSV	–	Comma-separated values
DBMS	–	Database management systems
DB	–	Database
DSN	–	Data Source Name
GUI	–	Graphical user interface
HTML	–	Hypertext Markup Language
HTTPS	–	Hypertext Transfer Protocol Secure
HTTP	–	Hypertext Transfer Protocol
IAX	–	Inter-Asterisk eXchange
IDE	–	Integrated Development Environment
ID	–	Identification
IETF	–	Internet Engineering Task Force
IPv4	–	Internet Protocol version 4
IPv6	–	Internet Protocol version 6
IP	–	Internet Protocol
ITSP	–	Internet telephony service provider
IVR	–	Interactive Voice Response
JDBC	–	Java Database Connectivity
JS	–	JavaScript
NAT	–	Network address translation
ODBC	–	Open Database Connectivity
OS	–	Operating System
PBX	–	Private branch exchange
PSTN	–	Public switched telephone network
RDBMS	–	Relational database management system
RDNIS	–	Redirected Dialed Number Information Service
RFC	–	Request for Comments
RTP	–	Real-time Transport Protocol

SIP	–	Session Initiation Protocol
SQL	–	Structured Query Language
TCP	–	Transmission Control Protocol
UDP	–	User Datagram Protocol
UFW	–	Uncomplicated Firewall
URI	–	Uniform Resource Identifier
VM	–	Virtual Machine
VoIP	–	Voice over IP

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

Úvod

Internetová telefonie, také známá jako VoIP je v dnešní době pro komunikaci v reálném čase nejvíce rozšířená. Většina z nás ji využívá denně a možností je dnes opravdu mnoho. Technologii VoIP mohou také využívat call centra nebo organizace se zaměřením na hlasovou komunikaci nebo přenos multimédií. Organizace s tímto zaměřením budou také využívat monitorovací aplikace pro získaní informací o nabízených službách.

Cílem této bakalářské práce je realizace monitorovacího systému pro využití v call centrech k zobrazení informací jako informace o probíhajících hovorech, dostupné linky, počet volajících apod. Implementace systému telefonní ústředny Asterisk pro vytvoření prostředí menšího call centra, jeho propojení s databází a následná realizace webové aplikace, která na základě dat z databáze bude schopna monitorovat dění v systému Asterisk.

V teoretické části práce se seznámíme s technologií VoIP a protokoly, které využívá - především protokol pro signalizaci SIP. Poté následuje seznámení se systémem Asterisk PBX a jeho funkcemi. Hlavní důraz je kladen na jeho funkci **CEL** (Channel Event Logging), která slouží k vytváření záznamů. Seznámíme se také s funkcemi systému Asterisk, které jsou využity při realizaci prostředí menšího call centra a dále také s možnostmi integrace Asterisku s databázovými systémy. Na konec se v teoretické části obeznámíme s pojmy webový server a webová aplikace, jejich funkcemi a možnostmi využití dat z databází.

Praktická část práce je v první část zaměřena na instalaci a konfiguraci systému Asterisk pro vytvoření prostředí call centra a konfiguraci funkce CEL. Další část realizace je zaměřena na nastavení konektivity mezi Asteriskem a databázovým systémem MySQL pomocí konektoru ODBC, následné propojení výstupu záznamů CEL s databází. Poté následuje implementace webového serveru Apache Tomcat a realizace webové aplikace pro monitorování se zaměřením především na data, která jsou databází využívána pro monitoring. Poslední část obsahuje zobrazení zpracovaných dat v GUI webové aplikace a zhodnocení.

1 Voice over IP

Voice over IP (**VoIP**), také známá pod názvem internetová telefonie je skupina technologií pro hlasovou komunikaci a přenos multimédií pomocí protokolu IP. Z komunikačních technologií, které zprostředkovávají přenos hlasu je VoIP v dnešní době nejrozšířenější. Pro přenos hlasu využívá VoIP na třetí vrstvě modelu OSI protokol IP a na vrstvě čtvrté protokol UDP. Mezi základní prvky k uskutečnění hovoru patří dvě koncová zařízení a médium k přenosu dat mezi nimi.[1][2]

Technologie VoIP obsahuje řadu implementací lišících se podle standardu použitého k navázání spojení. V současné době jsou nejvíce využívány **SIP**, H.323, který je však na ústupu nebo také IAX, který je používán systémy Asterisk.[1]

Klasická veřejná telefonní síť, také známá pod zkratkou PSTN (public switched telephone network) byla vytvořena primárně k přenosu hlasu. PSTN funguje na principu fixního propojení dvou koncových zařízení, které mezi sebou v daný čas komunikují (probíhá hovor). Při spojení obou stran mezi sebou, veškerá dostupná přenosová kapacita je vyhrazena pouze pro daný telefonní hovor. Tento způsob zajišťuje velmi vysokou kvalitu a spolehlivost. Jedna z hlavních nevýhod oproti technologii VoIP je dostupnost, protože síť PSTN je pevně položená.[2]

Internetová telefonie na rozdíl od veřejné telefonní sítě pracuje na odlišném principu. Informace jsou přenášeny klasickými počítačovými sítěmi, kde hlavním principem není přenos hlasu pro hovor, ale přenos dat neboli paketů. Díky tomu, můžeme využívat souběžně s hovorem další služby nebo aplikace. Další výhodou poté může být dostupnost technologie VoIP, kde jediné co je prakticky potřeba, je připojení k internetu a to je dnes prakticky možné kdekoliv. Na druhou stranu, jedna z nevýhod je například kvalita přenosu hlasu, kde díky přenosu datagramů pomocí protokolu UDP není zaručené jejich doručení do cíle. Toto může být způsobeno např. zvýšenou latencí.[2]

Možnosti koncových zařízení, které můžeme použít jsou např. hardwarové IP telefony nebo softwarové řešení aplikaci na počítači nebo telefonu (tzv. softphone). Dostupné máme také služby, které využívají uzavřené sítě jen mezi sebou. Mezi ně patří třeba Discord, Skype nebo TeamSpeak. Tyto služby jsou v dnešní době z VoIP technologií velmi populární. Jako médium pro přenos dat mezi zařízeními se používá převážně Ethernet, Wi-Fi nebo mobilní data.

1.1 Internet Protocol (IP)

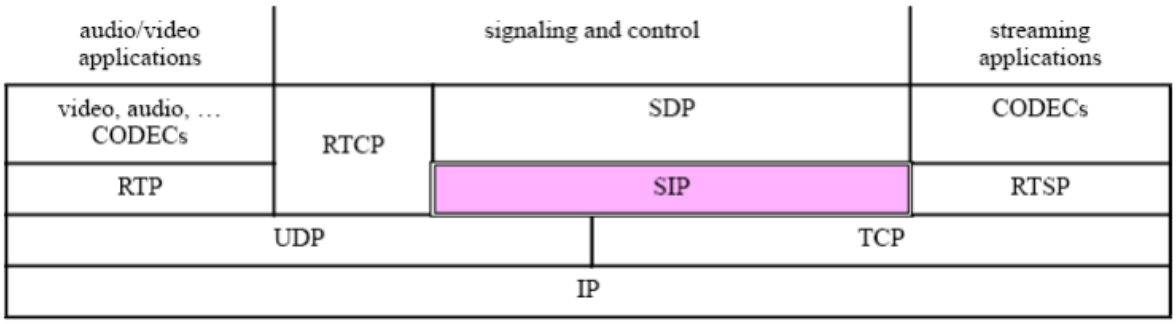
Internetový protokol (IP) je hlavní komunikační protokol v sadě internetových protokolů pro předávání datagramů v počítačové síti. Jeho směrovací funkce propojuje dohromady internetové sítě a v podstatě vytváří internet jako celek.

Úkolem IP je doručování paketů ze zdrojového do cílového hostitele na základě použití IP adres v hlavičkách přenášených paketů. Internetový protokol definuje strukturu paketů, které zapouzdřují přenášená data. Definuje také metody adresování, sloužících k označení datagramu zdrojovými a cílovými informacemi.[3]

Sám o sobě protokol neposkytuje žádnou záruku pro přenos dat a rozlišuje pouze pomocí IP adresy jednotlivých rozhraní v síti. První hlavní verzí protokolu IP byla verze 4 (IPv4), která je stále dominantní oproti verzi 6 (IPv6).

1.2 Session Initiation Protocol (SIP)

Session Initiation Protocol (**SIP**) je signalizační protokol používán v internetové telefonii (VoIP). SIP je využíván pro navázaní, modifikaci nebo ukončení relace v reálném čase. To zahrnuje multimediální relace v aplikacích pro přenos zvuku, videa a zpráv pomocí IP. Nejčastěji se však používá pro audio. Protokol SIP normálně pracuje na portu 5060/UDP, ale může fungovat i na 5060/TCP. První verze SIP protokolu byla uvedena v dokumentu RFC 2543, který byl vyvinut organizací IETF (Internet Engineering Task Force). Aktuální verzi popisuje dokument RFC3261.[4]



Obrázek 1: SIP protokol stack [5]

V síti jsou SIP entity identifikovány pomocí URI (Uniform Resource Identifier). URI se uvádí v níže uvedeném formátu.

```
sip:<jméno>@<doména/IP adresa>
```

URI také může obsahovat další údaje, jako např. číslo portu nebo použitý protokol na transportní vrstvě.[6][7]

Mezi vlastnosti protokolu SIP patří zejména to, že je velmi podobný protokolu HTTP (Hypertext Transfer Protocol), takže je čistě textově orientovaný. Díky tomu je protokol SIP velice odlehčený a zjednodušený oproti doporučení H.323, který byl dlouhou dobu standardem pro signalizaci v internetové telefonii.[4]

K tomu, aby protokol SIP zajistil VoIP spojení spolupracuje i s ostatními protokoly. Prvním z protokolů je protokol **SDP** (Session Description Protocol), který nám popisuje vlastnosti relace, jako typ média, použité kodeky nebo transportní protokol. Protokol SDP je umístěn v těle SIP paketů. Samotný přenos proudu audio a video dat mezi koncovými zařízeními je typicky realizován pomocí protokolu RTP (Real-time Transport Protocol).

1.2.1 Zprávy v SIP

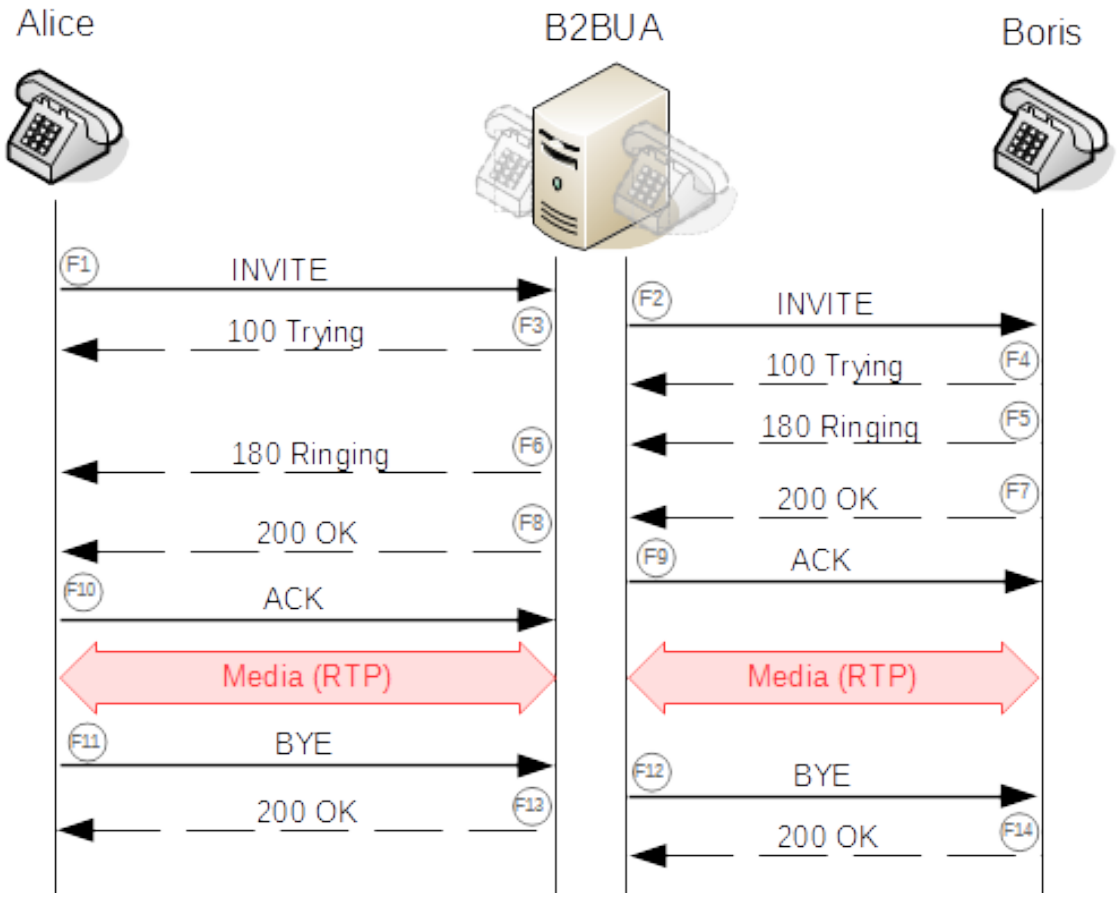
Komunikace u SIP probíhá primárně pomocí zasílání zpráv, které rozdělujeme do dvou kategorií. První kategorií jsou **žádosti**, neboli metody, které klient zasílá. Druhou kategorií jsou **odpovědi** ze serveru, podobně jako u protokolu HTTP. Odpověď taktéž jak u HTTP mají číslicový stovkový formát. Nejčastější příklady odpovědí jsou např. 100 Trying, 200 OK nebo 180 Ringing. Rozdělení zasílaných odpovědí je následující

- **1xx - průběh** - informační zpráva o průběhu relace
- **2xx - úspěch** - bezproblémové ukončení kroku
- **3xx - přesměrování** - je nutné přesměrování k dokončení požadavku
- **4xx - chyba klienta** - požadavek nelze dokončit, protože je chybný
- **5xx - serverová chyba** - server kvůli chybě nemůže zpracovat požadavek
- **6xx - globální chyba** - požadavek nelze nijak zpracovat, fatální chyba

Žádosti jsou převážně využívány k započetí určitého procesu, což může být např. sestavení nebo ukončení relace. V protokolu SIP máme definováno šest základních metod. Těmito metodami jsou REGISTER, INVITE, ACK, BYE, CANCEL a OPTIONS.[4] Popis těchto metod je uveden v tabulce č. 1.

Metoda	Popis
REGISTER	registrace URI na serveru
INVITE	sestavení nové relace
ACK	potvrzení sestavení relace
BYE	ukončení aktuální relace
CANCEL	přerušení žádosti o zahájení relace
OPTIONS	informace o dostupných možnostech

Tabulka 1: Základní SIP metody



Obrázek 2: Ukázka zasílání zpráv v protokolu SIP [8]

Session Description Protocol (SDP)

Session Description Protocol (SDP) popisuje parametry relace a je úzce spojen se signalizací SIP, se kterou je používán většinou dohromady a je to také textový protokol. Protokol SDP je rozdělen do tří hlavních skupin, které popisují vlastnosti spojení.[4]

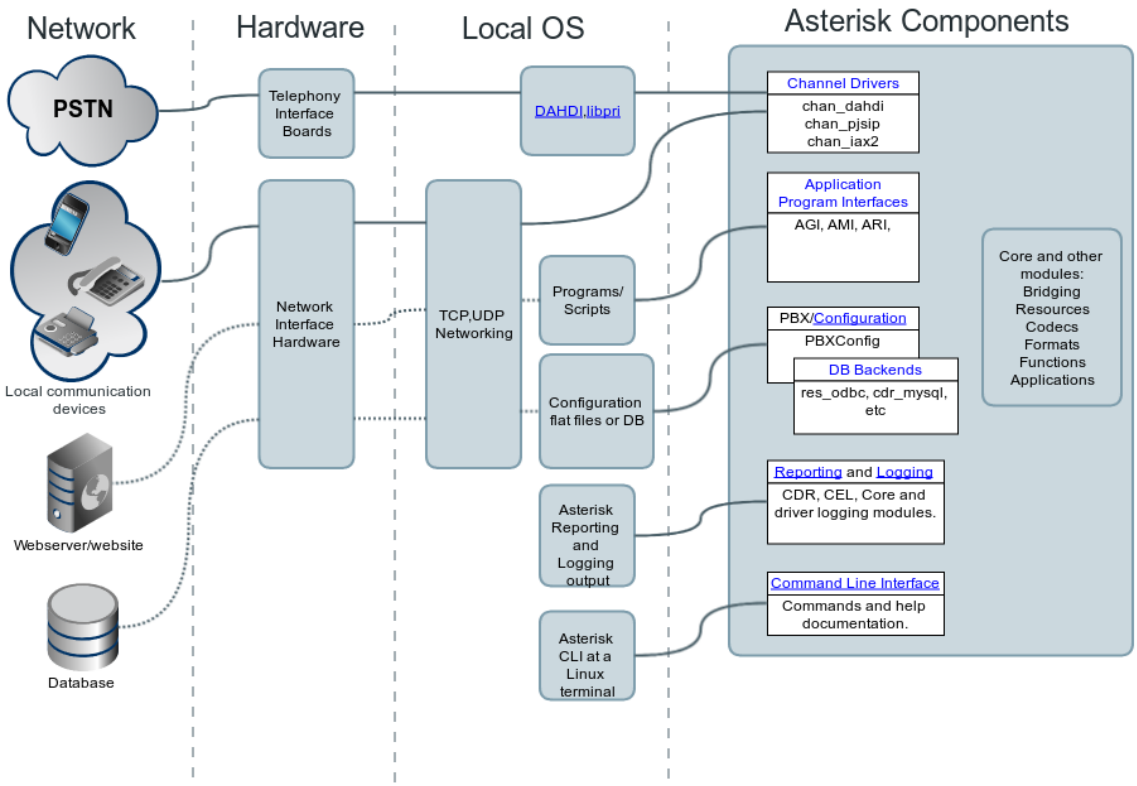
První skupina je *Session description*, která převážně popisuje vlastnosti relace a je nejrozsáhlejší. Obsahuje např. verzi protokolu, identifikátor relace, informace o relaci, URI, telefonní číslo nebo emailovou adresa. Druhou skupinou je *Time description*, která nám uvádí aktivní dobu relace a počet opakování. Poslední části je *Media description*, který popisuje informace o médiích v relaci jako je název a transportní adresa, označení nebo informace o spojení.

2 Asterisk

Asterisk je open source softwarová implementace telefonní ústředny (PBX), vyvíjena společností Digium, využívající technologie internetové telefonie (VoIP) s možností realizace na běžném počítači.

Systém Asterisk je vyvíjen od roku 1999. Původně začínal jako telefonní systém pro malé společnosti, ale o deset let později, co byl původně vydán se z Asterisku stal univerzální nástroj pro budování aplikací pro komunikační účely. Dnes Asterisk může být použit k realizaci velkého množství systému pro komunikaci v reálném čase. Mezi tyto systémy patří VoIP brány pro mnohé protokoly (SIP, H.323, IAX), systémy pro call centra, SIP trunky, servery pro hlasové schránky, systém interaktivního hlasového průvodce (IVR) a mnoho dalších možností, které by jsme od telefonní ústředny mohli požadovat. Systém Asterisk je provozován na platformách Linux/Unix a je napsán primárně v jazyce C.[9]

Architektura Asterisku se skládá z komponentů, které mezi sebou komunikují (viz obrázek č.3. Asterisk typicky také spolupracuje s dalšími prvky v síti pro optimální funkčnost jako jsou databáze, koncová zařízení pro komunikaci, web servery nebo propojení se sítí PSTN.



Obrázek 3: Architektura systému Asterisku [10]

Základní komponenty systému Asterisk jsou ovladače kanálů, mezi které patří konfigurace kanálů v příslušných technologiích (např. SIP nebo IAX). Komunikace mezi koncovými zařízeními v systému Asterisk probíhá pomocí logických kanálů, které jsou vytvářeny Asteriskem k propojení s koncovými zařízeními. Kanály po ukončení komunikace zanikají. Dále mezi komponenty patří také samotná konfigurace vlastností PBX, konektivita Asterisku s databází, API možnosti jako rozhraní AMI nebo systémy pro vytváření záznamů. Mezi systémy pro tvorbu záznamů patří především funkce **CEL**, která je popsána v kapitole č.2.3.

Většina konfigurace komponentů systému Asterisk probíhá v adresáři /etc/asterisk, kde se nachází samotné konfigurační soubory.

2.1 Dialplán

Dialplán patří mezi nejdůležitější části systému Asterisk. Slouží k určování, kde se kdo dovolá při vytočení určitého čísla, přepojování hovorů, realizace IVR systému apod. Jednoduše řečeno nám určuje, jak systém Asterisk zpracovává hovory. Délka posloupnosti čísel v dialplánu pro vytáčení volajícím může být libovolně dlouhá. Z praxe toto známe například ze spojení s našim operátorem a pro spojení s druhým koncovým uživatelem.[11]

V Asterisku toto nastavení probíhá v konfiguračním souboru **extensions.conf** v adresáři /etc/asterisk. Definuje se zde chování, jak Asterisk pracuje s hovory. Základní syntaxe, která je použita v dialplánu se skládá ze dvou částí a to uvedení kontextu, který rozděluje dialplán do oddělených sekcí a samotných příkazů.[12] Názorná ukázka syntaxe v Asterisku je uvedena níže.

```
[<název kontext>]
exten => <číslo>,<priorita>,<aplikace>(<parametry>)
```

Detailnější příklad chování příkazů v konfiguraci dialplánu, spolu s podrobným popisem funkčnosti kontextů a názornou praktickou ukázkou najdeme v sekci 2.1.2 popisující kontexty.

2.1.1 SIP uživatel

Nastavení SIP uživatele v našem případě znamená vytvoření SIP kanálu pomocí kterých poté komunikují koncová zařízení. Tato konfigurace je definována v souboru **sip.conf** nebo novějším pjsip.conf, který je dostupný od verze 12. Uvedené konfigurační soubory nalezneme v adresáři s ostatní konfigurací Asterisku (/etc/asterisk).[13]

Po otevření konfiguračního souboru si můžeme prohlédnout uvedené možné příklady konfigurace, které jsou zde již uvedeny. Jednoduchá praktická ukázka pro vytvoření nového SIP kanálu v souboru sip.conf pojmenovaného 1000, kterým se bude moci uživatel přihlašovat ke koncovým zařízením může vypadat následovně.

```
[general]
context=example
bindaddr=192.168.1.100
allowguest=no
nat=no
```

```
[1000]
secret=password
type=friend
host=dynamic
qualify=yes
context=local
disallow=all
allow=ulaw
```

V konfiguraci z ukázky můžeme vidět několik základních nastavení v sekci general, která jsou platná pro všechny další uvedené kanály, které nepřepisují nastavení ve své konfiguraci. Dále zde máme konfiguraci pro SIP kanál s číslem 1000. Nyní si postupně rozebereme jednotlivá nastavení a co přesně v konfiguraci znamenají.[14]

- **context=<název>**: vyhrazuje kontext pro uvedeného SIP klienta v dialplánu, více se můžeme dočíst v sekci 2.1.2, která se zabývá kontexty
- **bindaddr=<IP adresa>**: IP adresa pro síťové rozhraní použité pro Asterisk
- **allowguest=<yes/no>**: povoluje nebo zakazuje anonymní volající do Asterisku
- **nat=<yes/no>**: uvádí, zda uživatel využívá NAT, při volbě yes uživatel NAT využívat nemusí pro funkčnost
- **secret=<heslo>**: heslo, pomocí kterého se uživatel musí přihlásit na SIP server Asterisku
- **type=<peer/user/friend>**: možnost peer používáme pro odchozí hovory, možnost user pro příchozí a friend kombinuje obojí, čili odchozí i příchozí hovory
- **host=<IP adresa/dynamic>**: uvádí adresu, ze které se přihlašuje klient
- **qualify=<yes/no/čas v ms>**: zjišťuje, zda je klient dostupný, výchozí hodnota je jednou za 60 vteřin
- **disallow=<all>**: zakazuje všechny kodeky, které může určitý klient použít
- **allow<kodek>**: povoluje určitý kodek, používá se spolu v kombinaci s disallow=all

2.1.2 Kontext

Správné pochopení jak fungují kontexty pří konfiguraci je velmi klíčové pro správnou funkčnost dialplánu. Jednoduše řečeno kontext definuje skupinu vlastností v dialplánu, která je dostupná pro určitý kontext. Následuje jednoduchá ukázka jak fungují kontexty.

```
[100]
exten => 110,1,Dial(SIP/110,30)
exten => 110,n,Voicemail(110)
exten => 110,n,Hangup

[200]
exten => 210,1,Dial(SIP/210,30)
exten => 210,n,Voicemail(210)
exten => 210,n,Hangup

exten => 220,1,Dial(SIP/220,30)
exten => 220,n,Dial(SIP/110,30)
exten => 220,n,Hangup
```

Z ukázky vidíme, že máme v dialplánu uvedené dva kontexty [100] a [200] (Kontexty mohou být pojmenovány libovolně). Kontext [100] má nastavené volání pomocí aplikace Dial() na kanál SIP/110. Při vytočení čísla 110 se po 30 vteřinách vyzvánění přepojení hovor do hlasové schránky. Podobně je nakonfigurován i kontext [200] pro číslo 210 a kanál SIP/210, avšak má uvedené i číslo 220, na které, když se zavolá a kanál SIP/220 hovor nevyzvedne do 30 vteřin, tak je hovor přesměrován na kanál SIP/110, na který se jiným způsobem nedá dovolat, protože se nachází v jiném kontextu. Uživatelé registrovaní pod kanály SIP/210 a SIP/220 můžou navázat spojení mezi sebou díky stejnému kontextu.

2.1.3 Aplikace Dial()

Aplikace **Dial()** je základní aplikací Asterisku, umožňující navázání spojení mezi uživateli. Aplikace může mít až čtyři parametry a její syntax vypadá následovně.

```
exten => <číslo>,<priorita>,Dial()(<technologie/adresa>,<timeout>,
                                   <další možnosti>,<URI>)
```

První parametr nám uvádí použitou technologii pomocí které je hovor umožněn. Může se jednat např. SIP nebo IAX. V případě možnosti SIP se na místo adresy udává název SIP kanálu. Timeout parametr uvádí, jak dlouho se bude aplikace Dial() snažit o spojení s volanou stranou. Třetí parametr uvádí další dostupné možnosti aplikace. Může se jednat např. o změně vyzváněcí melodie volajícího. Parametr URI je využíván velmi zřídka. Jeho funkce je zasílání URI, které bude zobrazeno na zařízení volané strany pokud koncové zařízení podporuje tuto funkci.[15] Aplikaci Dial() mají v dialplánu přiřazeny všechny kanály, na které se dá dovolat vytočením příslušného čísla. Po vytočení čísla se aplikace snaží navázat spojení s volající stranou po dobu parametru timeout. Pokud se spojení neuskuteční, aplikace je ukončena.

2.1.4 Aplikace Background() a WaitExten()

Aplikace **Background()** přehraje zvukový záznam podobně jako aplikace Playback(), ale zároveň během přehrávání naslouchá např. uživatelskému vstupu. Poté se Asterisk snaží najít shodu se vstupem a aktuálním kontextem v dialplánu. Pokud se najde shoda mezi vstupem a číslem v kontextu, Asterisk přepojí hovor na příslušné číslo. Aplikace má jako parametr zvukový soubor pro přehrání.[9] Syntaxe vypadá následovně.

```
exten => <číslo>,<priorita,Background(<zvukový soubor>)
```

Problém, který se nám může vyskytnout u příkazu **Background()** je, že po přehrání námi vybraného souboru se nám hovor ukončí. Tomuto problému se dá jednoduše předejít použitím aplikace **WaitExten()**. Ve většině případů zavoláme aplikaci WaitExten() hned po příkazu Background(). Jako parametr můžeme nebo nemusíme uvést dobu po jakou se má čekat na uživatelský vstup v sekundách. Pokud parametr neuvedeme, bude použit interní timeout parametr.[9]

Pro akci, kdy budeme chtít, aby se nám po vytočení čísla 100 přehrál zvukový soubor example ve složce sounds a budeme čekat na vstup pro přepojení 10 vteřin, použijeme následující příklad.

```
exten => 100,1,Answer
exten => 100,n,Background(/sounds/example)
exten => 100,n,WaitExten(10)
```

2.1.5 Aplikace Goto()

Aplikace **Goto()** je použita pro přeposlání hovoru do jiné části v dialplánu. Syntax pro tento příkaz vypadá následovně.

```
exten => <číslo>,<priorita>,Goto(<kontext>,<číslo>,<priorita>)
```

Příkaz Goto() může být zavolán s jedním, dvěma nebo třemi parametry. Při použití příkazu s jedním parametrem, aplikace se přesune na uvedenou prioritu (parametr) v aktuálním čísle. Při zavolání aplikace se dvěma parametry, bude první parametr brán jako číslo a druhý jako priorita v aktuálním kontextu. Po použití všech tří parametrů se Asterisk přemístí do kontextu podle prvního parametru, na číslo z druhého parametru a prioritu z třetího parametru. Na ukázce je zobrazeno přesunutí při zavolání čísla 100 na číslo 200 v kontextu example a prioritu dvě.[15]

```
exten => 100,1,Goto(example,200,2)
```

Příkaz Goto() můžeme především využít při přesměrovávání do určitých IVR (Interactive Voice Response) systému z jiných kontextů.

2.2 Fronty

Pojem fronta je používán především v call centrech, který nám představuje řadu příchozích hovorů, čekajících na přepojení na člena fronty (agenta) zodpovědného za příslušnou frontu. Základní prvky fronty v Asterisku jsou následující

- Přidání příchozích hovorů do fronty
- Typ strategie, podle které se hovory přepojují na členy fronty
- Členové fronty (agenti), kteří odpovídají na hovory
- Hudba, která hraje volajícím při čekání

Konfigurace front, jako je např. název fronty, použitá strategie nebo statičtí agenti je definována v souboru **queues.conf** v adresáři /etc/asterisk.

Asterisk přesměruje příchozí hovory do front. Hovory jsou poté přepojovány na dostupné agenty podle specifické konfigurace každé fronty. Agenti jsou lidé, kteří odpovídají na příchozí hovory z front, ke kterým jsou přiřazeni.[16]

Člen fronty může být definován staticky jako agent v souboru agents.conf, nebo dynamicky jako SIP klient (např. SIP/agent100), kteří se v příslušných frontách přihlašují a odhlašují. Přihlašování a odhlašování ze specifických front pro konkrétní SIP uživatele musí být definováno v dialplánu.

Při konfiguraci se volající do front přidávají pomocí příkazu **Queue()**, který má jako první parametr název fronty, do které má být volající přidán. Příkaz Queue() může mít mnoho dalších parametrů jako např. různá nastavení, časový limit fronty nebo přidání na určitou pozici. Samotná tato konfigurace probíhá v souboru dialplánu, což je již zmíněný extensions.conf.

```
exten => 100,1,Queue(example)
```

Na výše uvedeném praktickém příkladu jsme v dialplánu nastavili, že po vytočení čísla 100 je volající zařazen do fronty s názvem example.

2.2.1 Přiřazování členů do fronty

Dynamické přiřazování členů se realizuje v konfiguračním souboru dialplánu **extensions.conf**. Pro přidání člena do fronty se požívá příkaz **AddQueueMember()**, kde je parametr příslušná fronta, do které chceme člena přidat. Pro odstranění použijeme příkaz **RemoveQueueMember()** opět s parametrem fronty. Ukázka konfigurace může vypadat následovně.

```
exten => 100,1,Answer
exten => 100,n,AddQueueMember(example)
exten => 100,n,Hangup

exten => 200,1,Answer
exten => 200,n,AddQueueMember(example)
exten => 200,n,Hangup
```

Po zavolání na číslo 100, bude volají číslo přiřazeno jako člen do fronty example a při zavolání čísla 200, bude vyřazeno z fronty example.

2.2.2 Strategie

Strategie ve frontě nám udává, jak budou hovory distribuovány mezi členy fronty.[16] Je definována v souboru **queues.conf** a možnosti jsou následující

- ringall - zavolá všem dostupným členům (výchozí možnost)
- random - volá náhodně
- rrmemory - volá členům postupně, pamatuje si posledního člena
- leastrecent - zavolá na nejméně volaného člena fronty
- fewestcalls - zavolá člena s nejméně dokončenými hovory

2.2.3 Interactive Voice Response

Interactive Voice Response (**IVR**) je technologie, která automatizuje rutinní záležitosti, jako např. interakce se zákaznickým servisem. To se realizuje tím, že se umožňuje volajícím komunikovat pomocí zadávání číslic na koncových zařízeních. Základním příkladem aplikace IVR je

hlasové menu nebo automatická obsluha. Volající jsou prezentováni nahrávkou, která jim uvede menu s možnostmi, na které volající reagují výběrem číslice.[17]

Jedním z hlavních přínosů systému IVR je úspora v porovnání s placením zaměstnanců, kteří by se museli zabývat neintuitivními a opakujícími se úkoly. Další z výhod tohoto systému jsou neomezené hodiny, kdy se dá na společnost používající korektně nakonfigurovaný IVR dovolat. Tímto se dají poskytnout výborné zákaznické služby přes víkend nebo mimo pracovní dobu.[17]

Automatizace těchto funkcí má smysl, pouze pokud jsou požadované úkoly poměrně jednoduché a dá se je snadno realizovat. Za příklady těchto úkolů mohou být považovány automatické odpovědi nebo přepojování do front pro spojení se zákaznickým servisem.

V Asterisku se systém IVR realizuje v konfiguračním souboru dialplánu **extensions.conf**, kde se využívají hlavně aplikace **Playback()** a **Background()** pro přehrávání zvukových záznamů volajícímu, dále aplikace **WaitExten()** pro naslouchání pro vstup ze strany volajícího a **Goto()** pro přesměrovávání do samotného IVR nebo dalšího sub menu. O těchto aplikacích více v kapitolách 2.1.5 a 2.1.4.

2.3 Channel Event Logging (CEL)

Channel Event Logging, neboli **CEL**, je systém pro logování údajů probíhajících během hovoru v Asterisku. Tento systém byl vytvořen za účelem pro vytvoření mnohem více detailních záznamů v průběhu relace a událostí probíhajících během ní. Systém CEL vytváří komplexní záznamy o událostech během relace již od vytvoření kanálu až po jeho terminaci, které nám poskytují velmi podrobný popis jak relace probíhala a různých detailů, které se během ní udály. Jedním z důvodu vytvoření systému CEL bylo nahrazení systému CDR, se kterým mohly nastat problémy při využití v aplikacích, které požadovaly větší důraz na velmi vysoký detail u záznamů. Využívat záznamy CEL mohou např. monitorovací aplikace, které vyžadují vysokou podrobnost o probíhajících událostech. Záznamy CEL jsou také vhodné pro realizaci systémů pro zpoplatnění služeb.[15][9]

Jeden záznam, neboli řádek v logu obsahuje ve výchozím nastavení mnoho informací. Podrobný popis všech položek je uveden v tabulce č.2. První položka v záznamu pod názvem **eventtype**, nám uvádí typ události, která proběhla. Tato hodnota o události je v celém záznamu nejdůležitější. Ostatní hodnoty popisují již informace, které nastaly u příslušné události, jako čas a datum, volané číslo nebo použitá aplikace. Nejčastější typy událostí a jejich popisy, se kterými se setkáme, nalezneme v tabulce č.3.

Dalo by se říci, že záznamy jsou rozděleny do skupin, kde každá skupina má přiřazené unikátní ID, hodnotu **linkedid**, která je identická po dobu trvání celé relace. V logu jednoduše poznáme, které záznamy patří k určitým relacím. Tato skupina obvykle začíná údajem s typem události s hodnotou **CHAN_START**, což znamená, že byl vytvořen nový kanál. V tomto okamžiku je přiřazeno nové unikátní ID **linkedid**. Tato situace nastává hned, když např. uživatel vytočí platné číslo pro určitý kanál. Hodnota události **CHAN_START** se také může objevit i v průběhu relace při vytvoření dalšího kanálu, jako je třeba přesměrování na agenta ve frontě. Při vyzvednutí hovoru a navázání přímého spojení nastávají události **ANSWER** a **BRIDGE_ENTER**, poté následuje samotná část hovoru. Po zavěšení následují události **BRIDGE_EXIT** a **HANGUP**, které nám oznamují, že hovor byl ukončen. Úplně nakonec se setkáme s událostmi **CHAN_END** a **LINKEDID_END**, kde **CHAN_END** oznamuje, že kanál zanikl a **LINKEDID_END** uzavírá celou relaci a je to poslední údaj s unikátním ID **linkedid**.

Položka	Popis
eventtype	typ události, viz tabulka č.3
eventtime	čas kdy nastala událost
cid_name	uvedené jméno u CID
cid_num	uvedené číslo u CID
cid_ani	hodnota ANI u CID
cid_rdnis	hodnota RDNIS u CID
cid_dnid	hodnota DNID u CID
exten	použité číslo v dialplánu
context	uvedený kontext v dialplánu
channame	přiřazený název kanálu pro příslušnou událost
appname	název použité aplikace
appdata	argument použitý příslušnou aplikací
amaflags	hodnota AMA pro událost, uživatelsky přiřazena
accountcode	přiřazený údaj o uživateli
peeraccount	přiřazený údaj o druhé straně spojení
uniqueid	přiřazené ID pro specifickou instanci kanálu
linkedid	ID přiřazené po celý hovor přes všechny události
userfield	přerušení žádosti o zahájení relace
peer	název kanálu druhé strany při přímém spojení

Tabulka 2: Všechny možné položky u funkce CEL

Událost	Popis
CHAN_START	vytvoření kanálu
CHAN_END	zaniknutí kanálu
ANSWER	vyzvednutí hovoru
HANGUP	zavěšení, obvykle ihned následuje CHAN_END
APP_START	spuštění sledované aplikace (musí být definována v cel.conf)
APP_END	ukončení sledované aplikace
PARK_START	odložení hovoru
PARK_END	zpětné převzetí hovoru
BRIDGE_ENTER	začátek přímého spojení, vyvolán aplikacemi jako Queue()
BRIDGE_EXIT	konec přímého spojení
BRIDGE_UPDATE	nastává při změně informací během přímého spojení kanálů
LINKEDID_END	zaniknutí posledního kanálu s příslušným linkedid

Tabulka 3: Typy událostí (eventtype) v systému CEL

2.3.1 Ukázka záznamů CEL

Na obrázku č.4, můžeme vidět, jak by mohla vypadat praktická ukázka záznamů **CEL**. První věc, kterou můžeme vyčíst je, že se jedná o jednu relaci, jelikož hodnota **linkedid** je stále stejná. Dále díky časovým záznamům vidíme, že tabulka je seřazena sestupně. U prvního záznamu s událostí **CHAN_START** si můžeme všimnout, že uživatel z kontextu user_group_a a číslem 1101 volal na číslo 911, kde byl přepojen systémem IVR (kontext ivr_user_menu) na kanál SIP/8101 z kontextu agent_group_a, který je členem ve frontě označené číslem 1 v dialplánu. Dále si můžeme všimnout událostí **ANSWER** a **BRIDGE_ENTER**, které uvádí vyzvednutí začátek samotného hovoru kanálem 8101, kde vidíme, že se jedná o frontu s označením sales. Časový rozdíl mezi událostmi **BRIDGE_ENTER** a **BRIDGE_EXIT** je aktuální délka hovoru. Poté vidíme, že hovor byl ukončen na straně s kanálem SIP/8101. Jako další události následují uzavření kanálu SIP/8101 (kanál pro komunikaci s agentem) a úplně prvního vytvořeného kanálu na SIP/1101 a ukončení celé relace **LIKNEDID_END**.

eventtype	eventtime	cid_num	exten	context	channame	appname	appdata	uniqueid	linkedid	peer
LINKEDID_END	2019-04-15 09:14:02	1101	1	ivr_user_menu	SIP/1101-0000000e			1555312401.96	1555312401.96	
CHAN_END	2019-04-15 09:14:02	1101	1	ivr_user_menu	SIP/1101-0000000e			1555312401.96	1555312401.96	
HANGUP	2019-04-15 09:14:02	1101	1	ivr_user_menu	SIP/1101-0000000e			1555312401.96	1555312401.96	
CHAN_END	2019-04-15 09:14:02		1	agent_group_a	SIP/8101-0000000f	AppQueue	(Outgoing Line)	1555312407.99	1555312401.96	
HANGUP	2019-04-15 09:14:02		1	agent_group_a	SIP/8101-0000000f	AppQueue	(Outgoing Line)	1555312407.99	1555312401.96	
BRIDGE_EXIT	2019-04-15 09:14:02	1101	1	ivr_user_menu	SIP/1101-0000000e	Queue	sales	1555312401.96	1555312401.96	
BRIDGE_EXIT	2019-04-15 09:14:02		1	agent_group_a	SIP/8101-0000000f	AppQueue	(Outgoing Line)	1555312407.99	1555312401.96	SIP/1101-0000000e
BRIDGE_ENTER	2019-04-15 09:13:30	1101	1	ivr_user_menu	SIP/1101-0000000e	Queue	sales	1555312401.96	1555312401.96	SIP/8101-0000000f
BRIDGE_ENTER	2019-04-15 09:13:30		1	agent_group_a	SIP/8101-0000000f	AppQueue	(Outgoing Line)	1555312407.99	1555312401.96	
ANSWER	2019-04-15 09:13:30		1	agent_group_a	SIP/8101-0000000f	AppQueue	(Outgoing Line)	1555312407.99	1555312401.96	
CHAN_START	2019-04-15 09:13:27		s	agent_group_a	SIP/8101-0000000f			1555312407.99	1555312401.96	
ANSWER	2019-04-15 09:13:21	1101	s	ivr_user_menu	SIP/1101-0000000e	Answer		1555312401.96	1555312401.96	
CHAN_START	2019-04-15 09:13:21	1101	911	user_group_a	SIP/1101-0000000e			1555312401.96	1555312401.96	

Obrázek 4: Praktická ukázka záznamů CEL

Hlavní nastavení funkce CEL probíhá v konfiguračním souboru **cel.conf**, který nalezneme v hlavním konfiguračním adresáři pro Asterisk /etc/asterisk. Možnosti, které se nám nabízí jsou

- **enable=<yes/no>** - zapíná nebo vypíná funkcí logování CEL
- **apps=<názvy aplikací/all>** - uvádí, které aplikace chceme v záznamech sledovat, možné aplikace jsou např. Queue(), při přesměrování do front nebo Dial()
- **event=<názvy událostí/all>** - uvádí, které typy událostí chceme zaznamenávat, pro seznam událostí viz tabulka č.3
- **dateformat=<datový formát>** - datový formát použitý v záznamech, možnosti formátů nalezneme ve zdroji [18]

Ukázka nastavení pro výstup záznamů CEL může vypadat následovně.

```
[general]
enable=yes
apps=Queue
events=ALL
dateformat = %F %T
```

Z uvedené ukázky konfigurace funkce CEL můžeme vidět, že logování CEL bylo pro Asterisk zapnuto, sledovaná je pouze aplikace Queue(), sledují se všechny události a datový formát je nastaven na "%F %T", který nám uvádí datum a čas ve formátu "2019-03-18 11:09:51". Asterisk ve výchozím nastavení ukládá záznamy logů do adresáře /var/log/asterisk, kde jsou ukládány ve formátu CSV.

Na následující ukázce na obrázku č.5 je náhled na záznam jednoho hovoru ve výchozím formátu CSV. V logu jsou data v jednotlivých záznamech oddělená čárkou a vždy začínají typem události.

```
"CHAN_START","2019-04-20 13:45:40","","1101","","","","911","user_group_a","SIP/1101-
00000001","","","3","","1555760740.7","1555760740.7",""
"ANSWER","2019-04-20 13:45:40","","1101","1101","","911","s","ivr_user_menu","SIP/1101-
00000001","Answer","","3","","1555760740.7","1555760740.7",""
"CHAN_START","2019-04-20 13:45:47","","","","","","s","agent_group_a","SIP/8101-
00000002","","","3","","1555760747.10","1555760740.7",""
"ANSWER","2019-04-20 13:45:50","","","","","","2","agent_group_a","SIP/8101-00000002","AppQueue","(Outgoing
Line)","3","","1555760747.10","1555760740.7",""
"BRIDGE_ENTER","2019-04-20 13:45:50","","","","","","2","agent_group_a","SIP/8101-00000002","AppQueue","(Outgoing
Line)","3","","1555760747.10","1555760740.7",""
"BRIDGE_ENTER","2019-04-20 13:45:50","","1101","1101","","911","2","ivr_user_menu","SIP/1101-
00000001","Queue","tech","3","","1555760740.7","1555760740.7","SIP/8101-00000002"
"BRIDGE_EXIT","2019-04-20 13:45:56","","","","","","2","agent_group_a","SIP/8101-00000002","AppQueue","(Outgoing
Line)","3","","1555760747.10","1555760740.7","SIP/1101-00000001"
"BRIDGE_EXIT","2019-04-20 13:45:56","","1101","1101","","911","2","ivr_user_menu","SIP/1101-
00000001","Queue","tech","3","","1555760740.7","1555760740.7",""
"HANGUP","2019-04-20 13:45:56","","1101","1101","","911","2","ivr_user_menu","SIP/1101-
00000001","","","3","","1555760740.7","1555760740.7",""
"CHAN_END","2019-04-20 13:45:56","","1101","1101","","911","2","ivr_user_menu","SIP/1101-
00000001","","","3","","1555760740.7","1555760740.7",""
"HANGUP","2019-04-20 13:45:56","","","","","","2","agent_group_a","SIP/8101-00000002","AppQueue","(Outgoing
Line)","3","","1555760747.10","1555760740.7",""
"CHAN_END","2019-04-20 13:45:56","","","","","","2","agent_group_a","SIP/8101-00000002","AppQueue","(Outgoing
Line)","3","","1555760747.10","1555760740.7",""
"LINKEDID_END","2019-04-20 13:45:56","","","","","","2","agent_group_a","SIP/8101-00000002","AppQueue","(Outgoing
Line)","3","","1555760747.10","1555760740.7",""
```

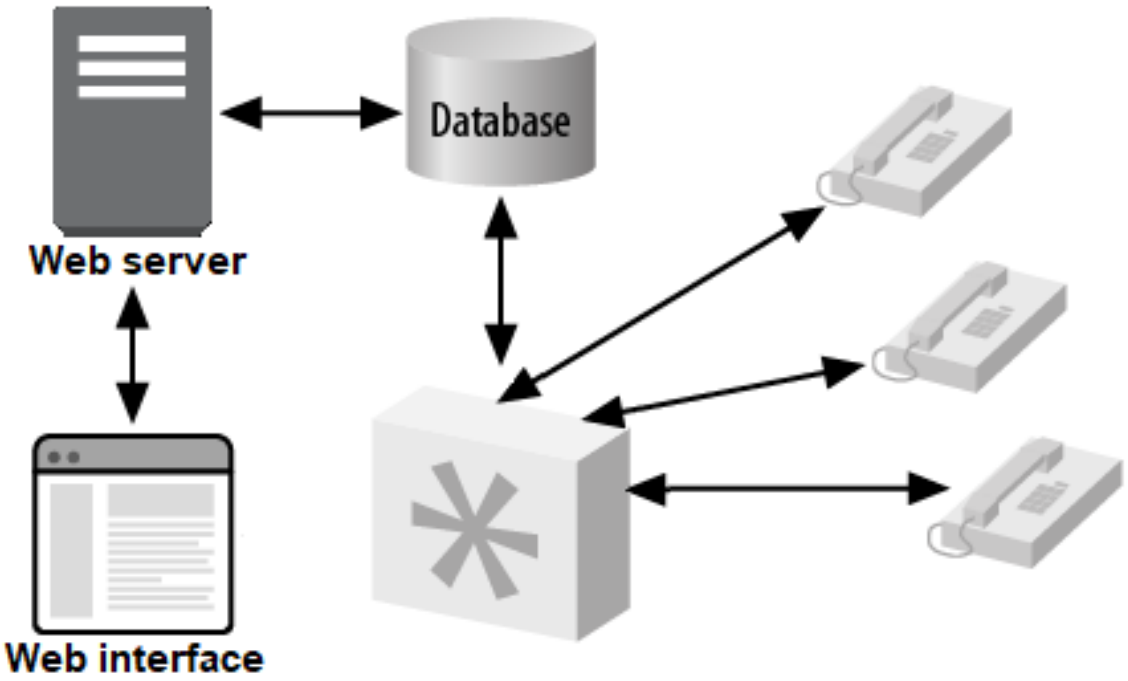
Obrázek 5: Ukázka záznamu CEL ve formátu CSV

2.4 Integrace s databází

Integrace Asterisku s databází přidává do systému mnoho možností. Jednou z hlavních výhod je možnost vybudování různých webových utilit, které nám mohou zjednodušit správu a monitorování systému Asterisk. Integrace s databází je velmi efektivní způsob pro přistupování dat, která mohou být zpracována jinými prostředky.[9][15]

Jako příklad můžeme uvést sledování hovorů v monitorovací aplikaci, ve které přesně vidíme průběh hovorů probíhajících v systému. Toho můžeme docílit pomocí integrace databáze s výstupem funkce pro logování **CEL**. To ale není jediné využití propojení Asterisku s databází. Mezi další využití integrace s databází, patří např. importování různých dat do systému Asterisk.[9][15]

Pro propojení Asterisku s databází se v systémech typu Linux nebo Unix běžně využívá konektor **ODBC**, který je blíž popsán v kapitole 2.4.1. Nejčastější typy systému pro řízení databází, se kterými se můžeme setkat jsou **MySQL**, což je systém pro řízení databází uplatňující relační databázový model. MySQL je vlastněn společností Oracle Corporation. Je to jeden z nejstarších databázových systémů a patří mezi nejrozšířenější. Je používán především programátory v prostředí zaměřeném na web development. Dalšími z rozšířených management systému jsou PostgreSQL nebo Microsoft SQL.



Obrázek 6: Integrace Asterisku s databází a webovým serverem [15]

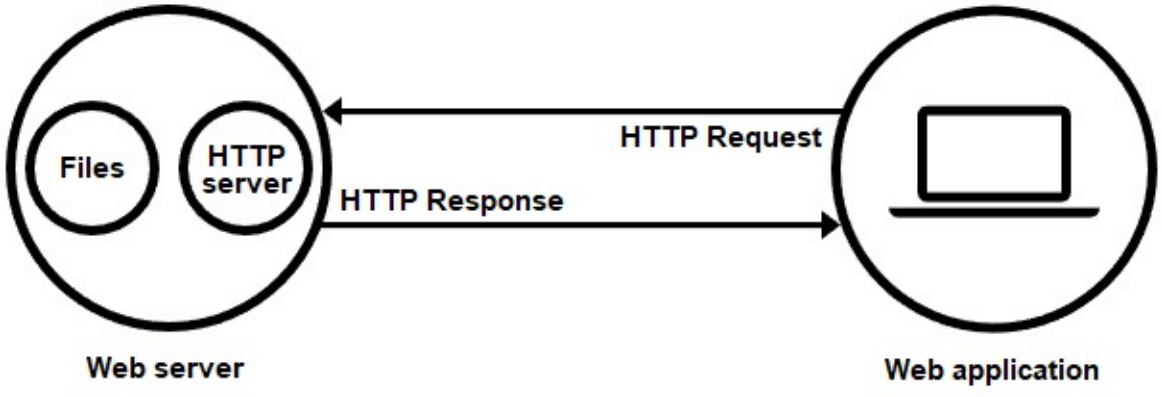
2.4.1 Open Database Connectivity

Open Database Connectivity (ODBC) byl původně vyvinut v 90. letech společnostmi Microsoft a Simba Technologies. ODBC je jedno ze standardních API (Application Programming Interface) pro přistupování k systémům pro správu databází (DBMS). Cílem při vývoji konektoru ODBC bylo vytvořit propojení s databází nezávisle na typu použitého DBMS nebo operačního systému. Aplikace využívají ODBC jako vrstvu pro konektivitu mezi databází. Aplikace mohou podstoupit změny v základní funkčnosti nebo úplně změnit platformu a to s minimálními nebo žádnými změnami v části kódu pro přístup k datům.[19]

Dnes je konektor ODBC stále velmi používán pro propojování aplikací s DBMS a má dostupnou verzi na většině platformách. Nicméně, s velkým nárůstem aplikací používajících HTML formát se snížila potřeba pro ODBC, protože většina aplikací založených na webových platformách se mohou připojit k požadované databázi (převážně MySQL) přímým spojením.

3 Webový server

Za webový server můžeme považovat software nebo hardware, určený pro běh zmíněného softwaru, který odpovídá na žádosti klientů. Webový server běžně obsahuje jednu nebo více webových stránek, které jsou uloženy na disku na straně serveru. Hlavní činností webového serveru je ukládat, zpracovávat a dodávat webové stránky klientům. Komunikace mezi serverem a klientem probíhá primárně pomocí **HTTP** protokolu. Dodané webové stránky jsou převážně **HTML** dokumenty, které většinou obsahují navíc kaskádové styly (CSS), skripty (JavaScript) nebo obrázky.[20]



Obrázek 7: Komunikace mezi klientem a serverem [20]

3.1 Hypertext Transfer Protocol (HTTP)

HTTP (Hypertext Transfer Protocol) je protokol určený ke komunikaci s webovými servery. HTTP protokol je čistě textově orientovaný a obvykle používá port 80/TCP. Protokol je založen na principu dotazů a odpovědí mezi klientem a serverem, obdobně jako u protokolu SIP (viz kapitola 1.2). Klient zasílá obvykle pomocí webového prohlížeče dotaz na server, obsahující informace jako označení požadovaného dokumentu, adresa serveru, informace o použitém prohlížeči apod. Server poté odpovídá na žádost klienta. Odpověď obsahuje informace, zda bylo možné nalézt dokument, typ dokumentu apod., za kterými následuje obsah samotného žádaného dokumentu.[21]

```
GET /example/asterisk HTTP/1.1
Host: asterisk.server.cz
User-Agent: Mozilla/4.00
```

Na výše uvedeném jednoduchém příkladu vidíme, jak může vypadat dotaz klienta. Dotaz obsahuje žádost o dokument /example/asterisk na serveru asterisk.server.cz a informaci o použitém webovém prohlížeči u klienta. Odpověď serveru může vypadat následovně.

```
HTTP/1.0 200 OK
Date: Thu, 14 Mar 2019 09:39:12 GMT
Server: Apache/1.3.27
Content-Type: text/html
<html>...</html>
```

Z ukázky odpovědí vidíme, že žádost byla úspěšná ve formátu 200 OK, stejně jako zmíněný protokol SIP. Odpověď udává datum a čas, použitý software serveru, typ formátu obsahu a samotný obsah HTML dokumentu.

3.2 Java Database Connectivity (JDBC)

Java Database Connectivity (JDBC) je jedno ze standardních API používaných v programovacím jazyce Java. JDBC je součástí Javy SE (Standard Edition) od JDK verze 1.1 z roku 1997. JDBC slouží jako vrstva mezi Java aplikací a databázovým systémem, která poskytuje aplikaci konektivitu s databází. Umožňuje aplikaci použití SQL příkazů jako dotazování (**SELECT**), úpravu (UPDATE) nebo vkládání (INSERT) do propojené databáze.[22]

Dotazovací příkaz SELECT slouží k získání dat z databáze. Příkaz vrací výsledek ve formátu tabulky, která obsahuje vybraná data.

3.3 Webová aplikace

Webová aplikace je rozhraní poskytované klientům pomocí webového serveru přes počítačovou síť. Webové aplikace jsou čím dál, tím víc populární díky jejich dostupnosti. Pro využívaní webové aplikace klient potřebuje pouze zařízení s webovým prohlížečem a internetové připojení.[23]

Obsah webové aplikace je primárně tvořen HTML dokumentem, společně s kombinací s kaskádovými styly (CSS) a skripty (JS). Dalšími soubory mohou být např. obrázky, audio nebo video. Tento obsah je uložen v adresáři, kde k němu přistupuje webový server.

3.3.1 HTML, CSS a JavaScript

HTML je jazyk pro tvorbu webových stránek. Skládá se z tzv. elementů, které jsou rozděleny na značku (tag), atributy a obsah. Každý element začíná vždy špičatou závorkou, následuje značka (např. h1 pro nadpis nebo img pro obrázek) a atributy. Poté následuje ukončení špičatou závorkou a samotný obsah. Element je ukončen značkou s lomítkem před ní ve špičatých závorkách. Ukázka pro vycentrovaný nadpis "Hello world!"může vypadat následovně.

```
<h1 align="center">Hello world!</h1>
```

Kaskádové styly (CSS) je jazyk, který se stará o vzhled elementů v HTML dokumentech. O Syntaxi CSS by se dalo říci, že se skládá ze selektorů s bloky. Selektor je samotný název pro

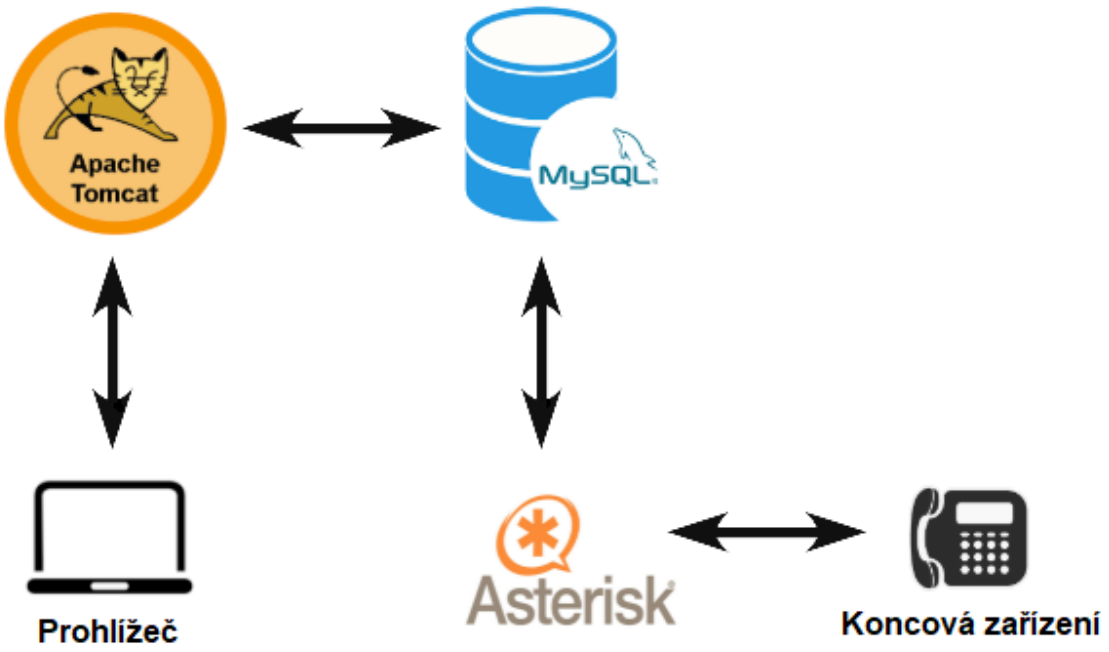
blok, který se poté přiřazuje elementům jako třída v HTML dokumentu. Za selektorem následuje blok, který je uzavřen ve složených závorkách a obsahuje vlastnosti pro dané elementy, jako šířka, výška, barva pozadí a mnoho dalších.

JavaScript (JS) je objektově orientovaný skriptovací jazyk. Jeho hlavní výhodou je zapisování přímo do HTML kódu. JavaScript je vykonáván pouze na straně klienta. Ve webových aplikacích se JavaScript stará např. o dynamický obsah nebo složitější funkčnost aplikace.

4 Rozvržení realizace projektu

První věc, kterou jsem před realizací práce provedl bylo rozvržení všech prvků, které budu využívat. Rozhodl jsem se pro využití databázového systému MySQL k ukládání dat ze záznamů CEL. Ke zprostředkování webového rozhraní pro monitorování jsem použil webový server **Apache Tomcat**. Přístup k databázi budou zajišťovat konektory JDBC ze strany webového serveru a ODBC ze strany Asterisku. JDBC a ODBC jsou API pro přístup k databázovým systémům. Díky tomuto řešení, bude systém nabízet možnosti pro rozšiřitelnost. Systém také není závislý na specifickém databázovém systému využívajících relační model, jelikož k přístupu k datům se využívá standardizovaný strukturovaný dotazovací jazyk SQL.

Provozování všech prvků jako MySQL serveru, webového serveru Apache Tomcat a Asterisku na jednom zařízení by hlavně v praxi nebylo úplně vhodné, ať už z důvodu škálovatelnosti nebo případného výpadku jednoho z prvků. Základní propojení mezi sebou je zobrazeno na obrázku č.8.



Obrázek 8: Propojení mezi jednotlivými prvky

Kompletní realizaci jsem rozvrhl do tří hlavních částí

- Instalace a konfigurace systému Asterisk jako call centrum
- Konfigurace MySQL serveru a jeho propojení se záznamy CEL z Asterisku
- Realizace webového serveru, který nám poskytuje webovou aplikaci pro monitorování

První z částí je systém Asterisk, který je provozován na virtuálním serveru. Druhá část realizace je MySQL server, který je taktéž provozován na virtuálním zařízení. MySQL běží na stejném virtuální stanici jako Asterisk. Z důvodu ušetření dostupných prostředků na zařízení provozující virtuální server. Dále také z praktických důvodu jako konfigurace systému Asterisk a MySQL serveru. Toto řešení by nebylo optimální. Ideálně by mělo být řešeno dvěma stanicemi. Poslední z částí je webový server Apache Tomcat, který běží na fyzickém počítači s OS Microsoft Windows 10 Pro 64-bit. K virtualizaci jsem použil nástroj Microsoft Hyper-V a jako systém pro virtuální servery byl využit Ubuntu 18.04. K testování na koncových zařízeních byly použity softphone aplikace Linphone.

Na obrázku č.9 je graficky znázorněna komunikace prvků mezi sebou v počítačové síti. IP adresy, které jsem v práci přiřadil zařízením jsem uvedl v tabulce č.4.

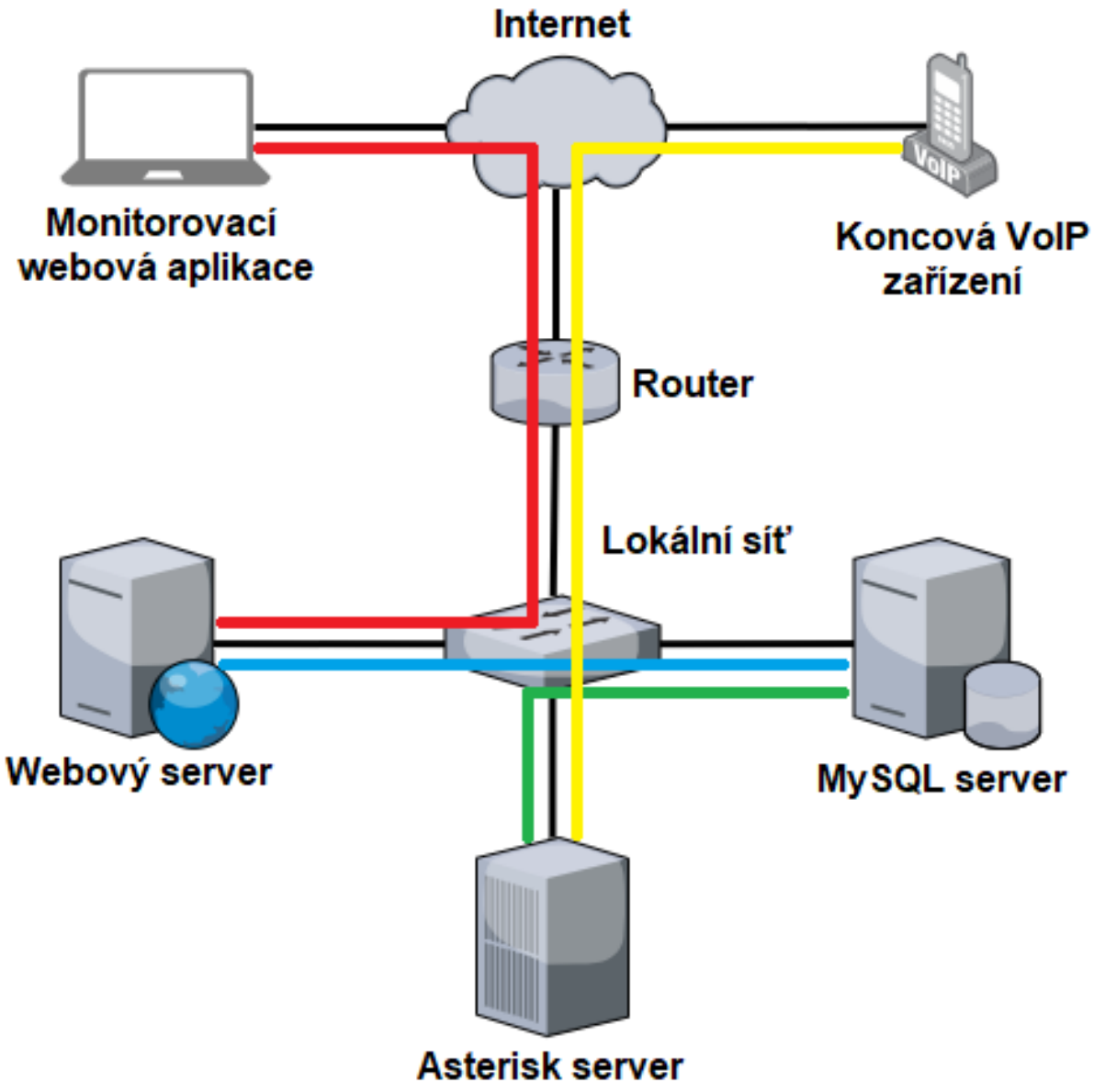
Pro správnou funkčnost projektu, je také nutno nastavit přesměrovávání portů v lokální sítí, které jsem pro mou realizaci uvedl v tabulce č.5. První dva řádky tabulky se starají o přesměrování portů protokolů SIP a RTP na adresu Asterisk serveru a třetí řádek se stará o přesměrování HTTP požadavků s výchozím portem na lokální port web serveru. Přesměrování je nutné pro přístup k webovému serveru a uskutečnění hovorů z internetu pomocí veřejné IP.

IP adresa	Popis
192.168.1.1	Výchozí brána
192.168.1.2	Fyzická stanice web serveru
192.168.1.34	Virtuální stanice MySQL serveru a Asterisku
81.200.53.38	Veřejná IP adresa

Tabulka 4: Použité IP adresy

Externí port	Interní port	Lokální IP adresa
5060	5060	192.168.1.34
10000-20000	10000-20000	192.168.1.34
80	8080	192.168.1.2

Tabulka 5: Přesměrování portů



Obrázek 9: Struktura projektu v počítačové síti

4.1 Použitý software

V této sekci je vypsán veškerý software, který jsem využil k realizaci této práce.

Asterisk 16.0.0

Pro více informací ohledně systému Asterisk viz sekce 2. Použil jsem Asterisk systém ve verzi 16.0.0, která byla nejaktuálnější při začátku realizace projektu.

MySQL Server 5.7.25

MySQL je široce využívaný databázový systém vyvíjen společností Oracle Corporation. K získání dat využívá jazyk SQL. Tento systém jsem využil k propojení výstupu záznamů CEL s databází a přístupu k datům ze strany web serveru.

Microsoft Hyper-V 10.0

Miscrosoft Hyper-V je virtualizační nástroj vyvíjen společností Microsoft, integrovaný přímo v OS Windows 10. Tento nástroj jsem využil ke správě virtuálního zařízení pro běh Asterisk a MySQL serveru.

Ubuntu 18.04 LTS

Ubuntu je volně dostupná Linuxová distribuce založená na operačním systému Debian. V projektu jsem Ubuntu využil pro chod MySQL serveru a Asterisk na virtuálním zařízení.

NetBeans IDE 8.2

NetBeans je volně dostupně integrované vývojové prostředí (IDE), které vlastní Oracle Corporation. NetBeans je primárně určen pro vývoj v jazyce Java. Toto IDE jsem zvolil pro realizaci webové aplikace kvůli podpoře serveru Apache Tomcat a jazyku Java.

Apache Tomcat 9.0.14

Apache Tomcat je webový server a servlet kontejner založen na jazyce Java. Je vyvíjen společností Apache Software Foundation. Tomcat jsem využil pro funkci web serveru pro monitorovací aplikaci.

Linphone

Linphone je softphone aplikace, kterou jsem využil k testování navrženého systému. Použil jsem aplikaci Linphone ve verzi pro Microsoft Windows, tak i verzi pro Android.

5 Instalace a konfigurace systému Asterisk

5.1 Instalace

Jak jsem již výše uvedl, Asterisk je nainstalován na virtuálním zařízení, kde běží Ubuntu 18.04. Zvolil jsem manuální instalaci nejaktuálnější verze Asterisk 16. Postup instalace systému Asterisk byl následující

1. Stažení nejaktuálnější dostupné verze Asterisku a následné rozbalení souboru

   ```
   sudo wget http://downloads.asterisk.org/pub/telephony/asterisk/
        asterisk-16-current.tar.gz
   sudo tar zxf asterisk-16-current.tar.gz
   ```

2. Přidání všech prerekvizit potřebných pro instalaci Asterisku

   ```
   sudo apt install wget build-essential subversion
   sudo contrib/scripts/get_mp3_source.sh
   sudo contrib/scripts/install_prereq install
   sudo wget --no-check-certificate https://raw.githubusercontent.com/
        asterisk/third-party/master/pjproject/2.8/pjproject-2.8.tar.bz2
   ```

3. Následné použití skriptu pro kontrolu všech potřebných prerekvizit

   ```
   sudo ./configure
   ```

4. V dalším kroku se provádí vybrání modulů, které mají být nainstalovány

   ```
   sudo make menuselect
   ```

   V mém případně jsem nechal vše ve výchozím nastavení, které pokrývá všechny mé potřeby.

5. Poté následuje kompilace a instalace Asterisku s příslušnými moduly a vytvoření základních konfiguračních souborů

   ```
   sudo make
   sudo make install
   sudo make basic-pbx
   ```

6. Jako finální věc po instalaci jsem provedl zpřístupnění portů 5060/UDP a 10000-20000/UDP ve firewallu serveru, které Asterisk používá pro protokoly SIP a RTP. K tomu jsem využil firewall službu UFW pomocí příkazů

   ```
   sudo ufw allow 5060/udp
   sudo ufw allow 10000:20000/udp
   ```

Nyní je systém Asterisk nainstalován a připraven ke konfiguraci. Jako první věc, kterou je potřeba po každé změně konfigurace Asteriku udělat, je znovu načtení konfiguračních souborů. K tomuto slouží příkaz

```
core reload
```

Uvedený příkaz je dostupný v rozhraní Asterisk CLI, do kterého se dá dostat pomocí příkazu

```
sudo asterisk -r
```

5.2 Konfigurace SIP kanálů

V této sekci je popsána konfigurace, kterou jsem provedl pro nastavení SIP kanálů, pomocí kterých se uživatelé u koncových zařízení přihlašují v systému Asterisk. Více informací obsahuje kapitola 2.1.1.

Konfigurační soubor pro nastavení SIP kanálu je **sip.conf** a nachází se v adresáři /etc/asterisk. Tato konfigurace je rozdělena do dvou hlavních části, a to sekce general, která obsahuje vlastnosti, které jsou aplikovány pro všechny ostatní sekce, pokud nejsou přepsány v dané sekci. Druhá část je deklarace skupin nebo samotných SIP kanálů. Kompletní konfigurace, kterou jsem použil je uvedena v sekci příloh A.1.

```
[general]
...
allowguest=no
externip=81.200.53.38
externaddr=81.200.53.38
localnet=192.168.0.0/255.255.0.0
```

Ve výše uvedeném výpisu ze sekce general jsem vybral důležité vlastnosti, které jsou stěžejní pro správnou funkčnost.

- **allowguest = no** - zakazuje anonymním volajícím připojení k systému Asterisk
- **externip a externaddr** - uvádí veřejnou IP adresu, která je použita uživateli, kteří se nepřipojují k Asterisku z lokální sítě, ale z internetu
- **localnet** - specifikuje adresu a masku pro použitou lokální síť, v mém případně 192.168.0.0/255.255.0.0

V sekci pro specifikaci používaných SIP kanálu, využívám možnosti pro deklaraci skupiny a následnou deklaraci specifických kanálů. Rozdělení v konfiguraci jsem provedl následovně

- Rozdělení kanálů pro běžné uživatele do dvou skupin, každá o čtyřech kanálech
  1. **user__group__a** s kanály 1101, 1102, 1103 a 1104
  2. **user__group__b** s kanály 1201, 1202, 1203 a 1204
- Rozdělení pro členy fronty (agenty), opět do dvou skupin o čtyřech kanálech
  1. **agent__group__a** s kanály 8101, 8102, 8103 a 8104
  2. **agent__group__b** s kanály 8201, 8202, 8203 a 8204

Z důvodu přehlednosti jsem rozdělil přiřazené kanály skupinám, kde číslo pro běžné uživatele začíná číslem 1 a následuje číslem skupiny. Čísla pro členy front začínají číslem 8 a opět následují číslem skupiny. Konfigurace těchto skupin vypadá následovně.

```
[agents_a](!)
type=friend
qualify=yes
nat=yes
host=dynamic
canreinvite=no
context=agent_group_a

[8101](agents_a)
secret=123

[8102](agents_a)
secret=123
...
```

Z uvedeného výpisu konfigurace je vidět, že se jedná o skupinu agents_a určenou pro členy fronty a jsou zde specifikovány kanály 8101 a 8102. Konfigurace pro ostatní skupiny a kanály vypadá obdobně. Nejpodstatnější vlastnosti konfigurace ve skupinách jsou

- **nat = yes** - udává, zda se uživatel využívající tento kanál nachází v síti za systémem NAT, jelikož chci umožnit volání přes internet volím možnost yes. V případě, že uživatel za NAT není a je zvolena možnost yes, tak to funkčnost nijak neomezuje
- **host = dynamic** - volím možnost dynamic, jelikož nevyžaduji specifickou IP adresu od koncových zařízení
- **context = agent_group_a** - udává, o jaký kontext se jedná, každá skupina má svůj příslušný kontext

5.3 Konfigurace front

Konfigurační soubor pro nastavení front **queues.conf** se nachází v adresáři /etc/asterisk. Detailní popis front a jejich možností v Asterisku je popsán v kapitole 2.2.

Obdobně jako tomu bylo při konfiguraci SIP kanálů, tak i zde je konfigurace rozdělena do dvou sekcí a to opět sekce general, která obsahuje nastavení sdílené pro všechny specifické fronty. Konfigurace front, kterou jsem v práci využil je vypsána v příloze A.2.

```
[general]
...
persistentmembers=yes
strategy=ringall
maxlen=0
joinempty=yes
```

V uvedené ukázce konfigurace ze sekce general, lze vidět několik důležitých vlastností front, které jsem použil. Tyto parametry zásadně ovlivňují jak fronty pracují mezi volajícími a členy fronty.

- **persistentmembers = yes** - zajišťuje, že dynamicky přihlášení členové fronty i po odpojení a znovupřipojení jejich zařízení nebo po restartu systému Asterisk, stále zůstávají členové příslušných front.
- **strategy = ringall** - uvádí, že bude využita strategie ringall pro volání členů, pokud níže uvedené fronty nemají uvedenou svou vlastní strategii volání
- **maxlen = 0** - maximální počet čekajících ve frontě, možnost 0 je použita pro žádné omezení
- **joinempty = yes** - udává možnost zařazení volajícího do fronty bez aktivních agentů nebo aktuálně nedostupnými agenty

Pro přidání samotných front do systému je stačí v konfiguračním souboru queues.conf pouze deklarovat. V mé práci využívám tři fronty, a to fronty **sales**, **tech** a **billing**, ke kterým se členové dynamicky přihlašují pomocí IVR systému. Členové front, kteří se budou přihlašovat využívají SIP kanály (specifikované v konfiguraci SIP kanálu) v rozmezí 8101 až 8104 a 8201 až 8204. Konfigurace samotných front vypadá následovně

```
[sales]
strategy=rrmemory

[tech]
strategy=leastrecent

[billing]
strategy=fewestcalls
```

Každé z front jsem taktéž přiřadil vlastní strategii pro volání členů front. Tyto strategie přepisují možnost ringall, která je uvedena v general sekci konfigurace.

5.4 Konfigurace dialplánu

V konfiguraci dialplánu se nachází základní nastavení jak Asterisk zpracovává hovory (viz kapitola 2.1. Konfigurace probíhá v souboru **extensions.conf**, který se nachází opět v adresáři s konfiguracemi /etc/asterisk. Kompletní konfigurace dialplánu je uvedena v příloze A.3.

Realizace dialplánu jsem rozdělil do dvou částí. V první z částí se nachází plán pro skupiny vytvořených linek pro běžné uživatele z konfigurace, kterou jsem provedl v sekci 5.2. Skupiny

jsou rozděleny dle jejich příslušných kontextů. Společně se v první části také nachází uživatelský IVR systém, který volající uživatele přepojuje do front, kterou si vybírají zmáčknutím číslice. Druhá část této konfigurace je obdobná s tím rozdílem, že se zde nachází plán pro členy fronty (agenty) a hlavně systém IVR, který umožňuje agentům se dynamicky přihlašovat a odhlašovat z front.

5.4.1 Dialplán pro uživatele

V této části především popisuji způsob implementace IVR systému pro běžné volající.

```
[user_group_a]
exten => 1101,1,Dial(SIP/1101,30)
exten => 1101,n,Hangup

exten => 1102,1,Dial(SIP/1102,30)
exten => 1102,n,Hangup

...

exten => 911,1,Goto(ivr_user_menu,s,1)
```

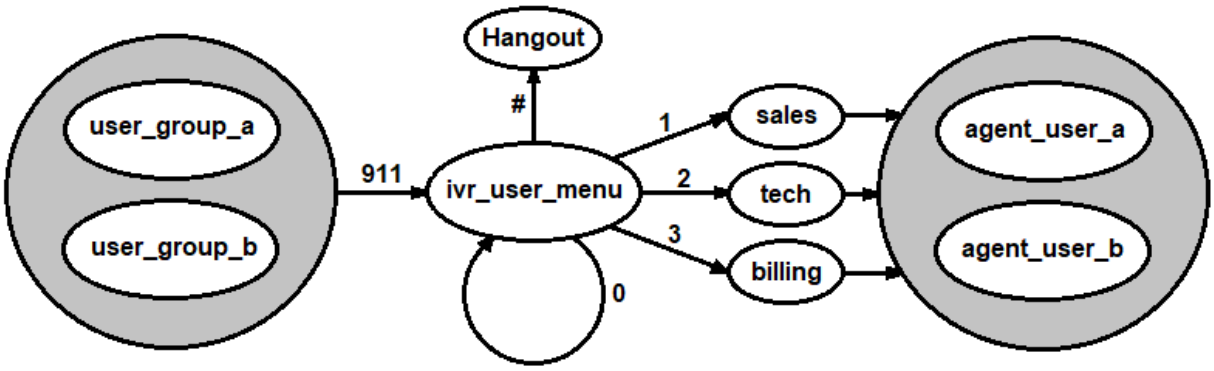
Ve výše uvedeném výpisu konfigurace má skupina uživatelů s kontextem *user_group_a* možnost přímého volání na čísla 1101 a 1102. Při vytočení těchto čísel se Astaerisk pokusí kontaktovat příslušný SIP kanál. Pokud volaný uživatel do třiceti vteřin hovor nevyzvedne, hovor ukončí. Stejnou konfiguraci obsahují i SIP linky 1103 a 1104, které nejsou v příkladu uvedeny. Stejným způsobem, jsem taktéž provedl konfiguraci pro uživatelskou skupinu *user_group_b*.

Jakmile uživatelé z kontextu *user_group_a* nebo *user_group_b* vytočí číslo 911 jsou pomocí aplikace Goto() přesměrováni do kontextu *ivr_user_menu* na pozici *s* a prioritu 1. Kontext *ivr_user_menu* obsahuje implementaci uživatelského IVR systému, jehož konfigurace je následující.

```
[ivr_user_menu]
exten => s,1,Answer
exten => s,2,Background(/var/lib/asterisk/sounds/ivr_promt_user)
exten => s,n,WaitExten

exten => 1,1,Playback(/var/lib/asterisk/sounds/sales_message)
exten => 1,n,Background(queue-callswaiting)
exten => 1,n,Queue(sales)

exten => 2,1,Playback(/var/lib/asterisk/sounds/tech_message)
exten => 2,n,Background(queue-callswaiting)
exten => 2,n,Queue(tech)

exten => 3,1,Playback(/var/lib/asterisk/sounds/billing_message)
exten => 3,n,Background(queue-callswaiting)
exten => 3,n,Queue(billing)

exten => 0,1,MusicOnHold(default)
exten => 0,n,Goto(ivr_user_menu,s,2)

exten => #,1,Playback(vm-goodbye)
exten => #,n,Hangup

exten => t,1,Goto(#,1)
exten => i,1,Playback(invalid)
```

Uživatelský IVR systém jsem realizoval následujícím způsobem. Po přesměrování uživatelů do kontextu uživatelského IVR jsou volající uvítání výzvou pro výběr fronty podle zmáčklé číslice, do které budou přesměrováni. Výběrem čísla jedna až tři, je volajícím přehráno oznámení o vybrané frontě a následné přesměrování do příslušné fronty. Volbou čísla nula se výzva znova opakuje. Zvolení symbolu # se přehraje záznam s rozloučením a hovor je ukončen. Při neprovedení žádné akce po nějakou dobu nebo špatné volby, hovor také končí.

Na obrázku č.10 jsem graficky znázornil výše uvedenou konfiguraci pro přesměrovávání volajících pomocí uživatelského systému IVR.



Obrázek 10: Grafické znázornění uživatelského systému IVR

5.4.2 Dialplán pro členy front

V konfiguraci dialplánu pro členy front jsem implementoval pouze IVR systém, pomocí kterého se členové dynamicky přihlašují a odhlašují z vybraných front.

```
[agent_group_a]
exten => 911,1,Goto(ivr_agent_menu,s,1)

[agent_group_b]
exten => 911,1,Goto(ivr_agent_menu,s,1)
```

Kontextech příslušící skupinám určených pro členy front obsahují přesměrování při vytočení čísla 911 na kontext *ivr_agent_menu* jako tomu bylo i v uživatelských kontextech. Kontext *ivr_agent_menu* obsahuje implementaci IVR systému pro přihlašování a odhlašování členů. Jeho konfigurace je uvedena níže.

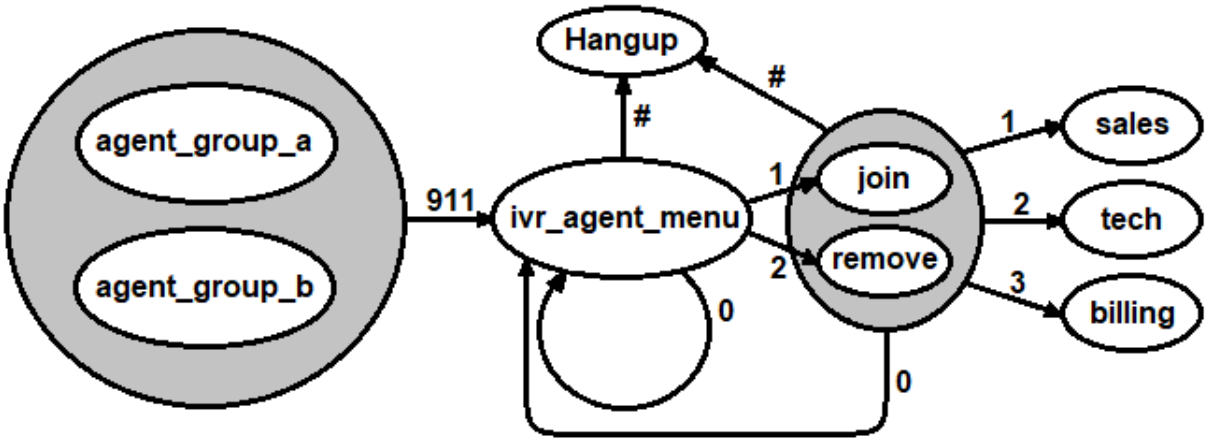
```
[ivr_agent_menu]
exten => s,1,Answer
exten => s,2,Background(agent-user)
exten => s,n,WaitExten

exten => 1,1,Playback(agent-loginok)
exten => 1,n,Goto(ivr_agent_login,s,1)

exten => 2,1,Playback(agent-loggedoff)
exten => 2,n,Goto(ivr_agent_logout,s,1)

exten => 0,1,MusicOnHold(default)
exten => 0,n,Goto(ivr_main,911,2)

exten => #,1,Playback(vm-goodbye)
exten => #,n,Hangup

exten => t,1,Goto(#,1)
exten => i,1,Playback(invalid)
```

Tento systém je základní strukturou shodný s uživatelským IVR systémem. To, co jsem zde pozměnil je, že po vybrání čísel 1 nebo 2 a po zvukové výzvě je agent přesměrován do dalšího menu IVR systému pomocí aplikace Goto(). Při vybrání čísla 1 je agent přesměrován do menu pro přihlašování k frontám a při volbě čísla 2 se jedná o menu pro odhlašování. Tato menu jsem nakonfiguroval následovně

```
[ivr_agent_login]
exten => s,1,Background(/var/lib/asterisk/sounds/ivr_promt_user)
exten => s,n,WaitExten

exten => 1,1,AddQueueMember(sales)
exten => 1,n,Playback(/var/lib/asterisk/sounds/sales_message)
exten => 1,n,Playback(agent-loginok)
exten => 1,n,Goto(#,1)

exten => 2,1,AddQueueMember(tech)
exten => 2,n,Playback(/var/lib/asterisk/sounds/tech_message)
exten => 2,n,Playback(agent-loginok)
exten => 2,n,Goto(#,1)
```

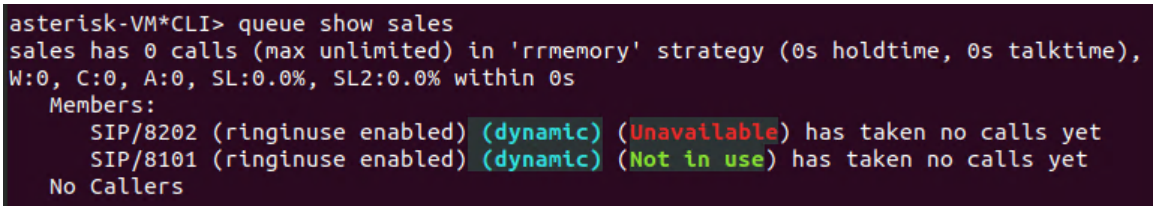
```
exten => 3,1,AddQueueMember(billing)
exten => 3,n,Playback(/var/lib/asterisk/sounds/billing_message)
exten => 3,n,Playback(agent-loginok)
exten => 3,n,Goto(#,1)

exten => 0,1,MusicOnHold(default)
exten => 0,n,Goto(ivr_agent_menu,s,2)

exten => #,1,Playback(vm-goodbye)
exten => #,n,Hangup

exten => t,1,Goto(#,1)
exten => i,1,Playback(invalid)
```

V ukázce se jedná o menu přihlašování agentů. Menu slouží pro přihlašování k frontám. Základy menu jsem opět realizoval obdobně jako u předešlých menu s tím rozdílem, že po výzvě k vybrání fronty si agent vybírá příslušnou frontu, ke které je poté přihlášen pomocí aplikace AddQueueMember(). Odhlašování agentů z front jsem realizoval totožným způsobem jako přihlašování. Jediným rozdílem je použitá aplikace RemoveQueueMember() namísto aplikace AddQueueMember(). Tento systém IVR jsem graficky znázornil na obrázku č.11.



Obrázek 11: Grafické znázornění systému IVR pro dynamické přihlašování agentů do front

Pro zobrazení informací o frontách jsem využíval příkaz *queues show* dostupný v Asterisk CLI. Ukázka možného výpisu informací v CLI ohledně fronty sales je zobrazena na obrázku č.12.



Obrázek 12: Funkce uživatelské IVR systému

5.5 Zapnutí funkce CEL

Funkce pro vytváření záznamů Channel Event Logging (CEL) je podrobně popsána v kapitole 2.3. Tato funkce je po instalaci systému Asterisk ve výchozím nastavení vypnutá. Konfigurační soubor s nastavením záznamů CEL **cel.conf** se nachází v adresáři /etc/asterisk. Ve výchozím nastavení jsou záznamy CEL ukládány pouze do souboru Master.csv v adresáři /var/log/asterisk/cel-custom.

Pro účely mé práce jsem v konfiguračním souboru funkce CEL nastavil typy událostí s aplikacemi, které budou sledovány a typ formátu data, který bude použit. Výsledná konfigurace funkce vypadá následovně.

```
[general]
enable=yes
apps=Queue,AddQueueMember,RemoveQueueMember
events=ALL
dateformat = %F %T
```

Význam jednotlivých použitých vlastností je následující

- **enable = yes** - zapíná samotnou funkci pro vytváření záznamů CEL
- **apps = Queue,AddQueueMember,RemoveQueueMember** - pro mé potřeby jsem nastavil aplikace, které budou funkcí CEL sledovány na aplikaci Queue(), pro sledování, kdy byli volající zařazení do front a aplikace AddQueueMember a RemoveQueueMember, pro monitorování aktivních agentů s příslušnými frontami
- **events = ALL** - zaznamenávání všech typů událostí
- **dateformat = %F %T** - používá formát *year-mm-dd hh:mm:ss* pro čas záznamu

Pro ověření konfigurace funkce CEL, jsem využil příkaz *cel show status*, který je dostupný v prostředí Asterisk CLI. Tento příkaz vypíše všechny události a aplikace sledované funkcí CEL.

6 Integrace systému Asterisk s MySQL databází

6.1 Instalace a konfigurace MySQL serveru

MySQL server je systém pro správu databázových systému, který využívám k propojení s výstupem záznamů CEL. Postup pro instalaci a základní konfiguraci MySQL serveru, který jsem použil byl následující

1. Stažení a instalace MySQL serveru pomocí příkazu

   ```
   sudo apt-get install mysql-server
   ```

2. Přihlášení do MySQL serveru root uživatelem

   ```
   sudo mysql -u root -p
   ```

3. Vytvoření uživatele pro správu Asterisk databáze pro web server. Následné vytvoření databáze a přidání práv vytvořenému uživateli. Toto jsem realizoval pomocí příkazů

   ```
   CREATE USER 'asterisk'@'%' IDENTIFIED BY 'password';
   CREATE DATABASE asterisk;
   GRANT ALL PRIVILEGES ON asterisk.* TO 'asterisk'@'%';
   FLUSH PRIVILEGES;
   ```

   Znak % za uživatelským jménem udává, že uživatel nemá specifickou host adresu. Příkaz GRANT ALL PRIVILEGES dává uživateli asterisk plnou kontrolu nad databází asterisk. Pro ověření, zda je uživatel vytvořen jsem využil příkaz

   ```
   SELECT User, Host FROM mysql.user;
   ```

4. Vytvoření tabulky v asterisk databázi pro výstup záznamů CEL

   ```
   CREATE TABLE cel (...);
   ```

   Celá tabulka se všemi proměnnými je uvedena v příloze B.1. Vytvořená tabulka musí obsahovat totožné položky, jako ty, které jsou na výstupu v CEL záznamech nakonfigurovaných v sekci 5.5.

5. Jako poslední věc je třeba povolit přístup k MySQL serveru z lokální sítě. Toho jsem docílil úpravou konfiguračního souboru mysqld.cnf, který se nachází v adresáři /etc/mysql/mysql.conf.d. V konfiguračním souboru jsem zakomentoval řádek *bind-address = 127.0.0.1*, který omezuje připojení k MySQL serveru pouze z adresy localhost.

6.2 Instalace a konfigurace ODBC

Tato sekce se skládá ze dvou částí. V první části je popsána instalace unixODBC a ODBC ovladače pro MySQL server. Druhá část popisuje vytvoření konektoru, následné propojení s funkcí CEL a verifikaci spojení.

6.2.1 Instalace unixODBC a ovladače pro MySQL

1. Stažení nejaktuálnější verze ODBC pro MySQL a následné rozbalení pomocí příkazů

```
sudo wget https://dev.mysql.com/get/Downloads/Connector-ODBC/8.0/
    mysql-connector-odbc-8.0.15-linux-ubuntu18.04-x86-64bit.tar.gz
sudo tar zxf mysql-connector-odbc*
```

2. Následuje stažení unixODBC, což je standard pro ODBC, především na platformách Linux/Unix, který je později použit pro navázání spojení s výstupem funkce CEL

```
sudo apt-get install unixodbc
```

3. Nyní by již měla být konfigurace ovladače pro MySQL automaticky hotova. Konfigurační soubor odbcinst.ini se nachází v adresáři /etc. Obsah souboru by měl vypadat následovně.

```
[MySQL]
Description = ODBC for MySQL
Driver = /usr/lib/x86_64-linux-gnu/odbc/libmyodbc8w.so
Setup = /usr/lib/x86_64-linux-gnu/odbc/libodbcmyS.so
FileUsage = 1
Pooling = Yes
CPTimeout = 120
```

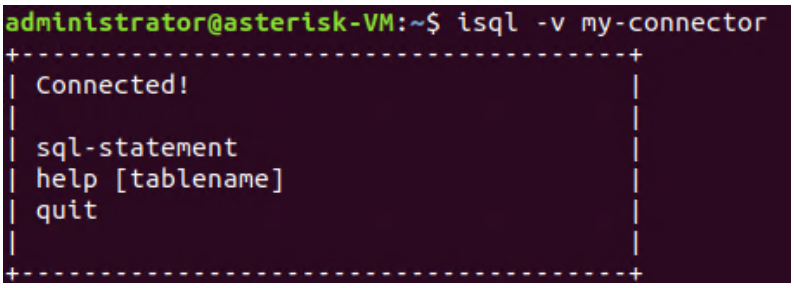
4. Jako poslední věc, kterou jsem provedl bylo zkopírování MySQL ODBC ovladače *libmyodbc8w.so* do adresáře /usr/lib/x86_64-linux-gnu/odbc (viz konfigurace v bodě č.3). Tento ovladač se nachází v adresáři lib ve dříve staženém a rozbaleném souboru s ODBC pro MySQL.

6.2.2 Vytvoření konektoru pro CEL

V souboru **odbc.ini** v adresáři /etc, se nachází konfigurace DSN (Data Source Name), které nám poskytují samotnou konektivitu k určité databázi pomocí ODBC ovladače. Ve výše uvedeném konfiguračním souboru jsem vytvořil nový DSN pro přístup k databázi *asterisk* pomocí MySQL uživatele *asterisk*.

```
[my-connector]
Description = MySQL connection to 'asterisk' database
Driver = MySQL
Database = asterisk
Server = localhost
User = asterisk
Password = asterisk
Port = 3306
Socket = /var/run/mysqld/mysqld.sock
```

Pro verifikaci vytvořeného DSN jsem využil příkaz *isql* (obrázek č.13) pomocí kterého, je možno si ověřit konektivitu s MySQL serverem pomocí ODBC.



Obrázek 13: Ověření konektivity s databází

Poslední věc, která zbývá pro propojení výstupu záznamů CEL s databází je specifikace DSN a tabulky přímo v konfiguraci Asterisku. To jsem provedl v konfiguračním souboru **cel_odbc.conf** v adresáři /etc/asterisk, který zajišťuje konektivitu mezi databází a funkcí CEL pomocí ODBC. Konfigurace pro mé vytvořené DSN vypadá následovně

```
[cel]
connection=asterisk
dsn=my-connector
username=asterisk
password=asterisk
table=cel
loguniqueid=yes
```

Nyní je třeba restartovat službu Asterisku, poté se budou všechny záznamy funkce CEL ukládat také do MySQL databáze.

7 Realizace webové aplikace

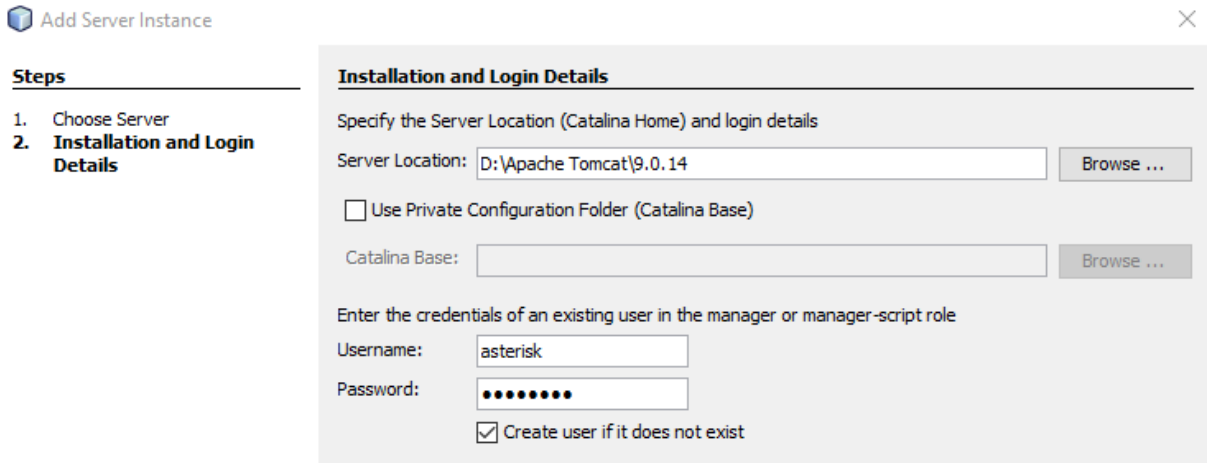
Aplikaci pro monitorování jsem realizoval pomocí nástroje NetBeans IDE 8.2 jako Java webovou aplikaci běžící na webovém serveru Apache Tomcat. Tuto realizace jsem provedel na fyzickém zařízení, na kterém běží OS Microsoft Windows 10 Pro. Realizaci aplikace jsem rozdělil do tří hlavních bodů

- Integrace webového serveru Apache Tomcat do nástroje NetBeans, vytvoření webové Java aplikace a nastavení konektivity s MySQL databází asterisk
- Způsob získávání potřebných informací pro monitorování ze záznamů CEL v databázi
- Zpracování získaných informací a jejich následné zobrazení ve webovém rozhraní

7.1 Webová Java aplikace

První věc, kterou jsem provedl před vytvořením nové Java web aplikace, bylo stažení webového serveru Apache Tomcat, který je dostupný na oficiálním webu organizace Apache Software Foundation. Následně jsem pokračoval těmito kroky

1. Přidání serveru Apache Tomcat do nástroje NetBeans. Tato možnost se nachází v sekci *Services* pod položkou *Add Server*. Následuje výběr serveru pro přidání (viz obrázek č.14).



Obrázek 14: Přidání serveru Apache Tomcat do NetBeans IDE

   Pro přidání je potřeba vybrat adresář se serverem. NetBeans v konfiguraci serveru vytvoří automaticky nového uživatele pro správu serveru.

2. Vytvoření nového projektu typu Java Web Application běžící na serveru Apache Tomcat.

3. Upravení souboru context.xml, nacházejícím se v adresáři META-INF, do kterého jsem přidal informace pro připojení k MySQL databázi asterisk pomocí JDBC, uvedené ve výpisu č.1. Kompletní výpis ze souboru context.xml je uveden v příloze C.9.

```
<Resource
    ...
    name="jdbc/asterisk"
    driverClassName="com.mysql.jdbc.Driver"
    username="asterisk"
    password="asterisk"
    url="jdbc:mysql://192.168.1.34:3306/asterisk?serverTimezone=Europe/
        Prague"
/>
```

Výpis 1: Připojení k databázi pomocí JDBC

Důležité informace jsou uvedení názvu databáze, jména a hesla pro přihlášení na MySQL server, typ použitého ovladače (v mém případě JDBC ovladač pro MySQL) a samotná adresa stanice provozující server.

4. Nutné je také přidání knihovny s JDBC ovladačem pro MySQL do projektu webové aplikace.

7.2 Způsob získávání informací ze záznamů CEL

Získávání dat pro monitorování, jsem realizoval pomocí několika metod. Jedná se o metody pro poskytnutí informací o probíhajících hovorech, nejvíce volané frontě, počet uskutečněných hovorů a počet volajících, kteří čekají ve frontě.

7.2.1 Metoda pro zjištění probíhajících hovorů

K získání dat o probíhajících hovorech jsem implementoval metodu **getAgentsOnCall**, její kompletní zdrojový kód je uveden v příloze C.1. Tato metoda vrací kolekci objektů, které reprezentují probíhající hovory mezi volajícími a členy fronty v Asterisku. Kolekce je realizována jako typ ArrayList, která obsahuje objekty s informacemi získanými ze záznamů CEL. Informace obsahují název kanálu agenta, název kanálu volajícího, název volané fronty a ID relace.

```
query = "SELECT channame, appdata, peer, linkedid "
  + "FROM cel "
  + "WHERE eventtype LIKE 'BRIDGE_ENTER' "
  + "AND appname LIKE 'Queue' "
  + "AND channame LIKE 'SIP/%' "
  + "AND peer LIKE 'SIP/%' "
  + "AND (eventtime > DATE_SUB(now(), INTERVAL 8 HOUR)) "
  + "ORDER BY id DESC;";
```

```
ResultSet rs1 = sh.getResultSet(query);
while(rs1.next()) {
    agentChanname = rs1.getString("peer").substring(4, 8);
    callerChanname = rs1.getString("channame").substring(4, 8);
    queueName = rs1.getString("appdata");
    linkedid = rs1.getString("linkedid");

    query = "SELECT linkedid FROM cel "
        + "WHERE eventtype LIKE 'BRIDGE_EXIT' "
        + "AND linkedid LIKE '" + linkedid + "' "
        + "AND(eventtime > DATE_SUB(now(), INTERVAL 8 HOUR)) "
        + "ORDER BY id DESC;";

    ResultSet rs2 = sh.getResultSet(query);
    boolean matchFound = false;
    while(rs2.next()) {
        if(rs2.getString("linkedid").equals(linkedid)) {
            matchFound = true;
        }
    }
    rs2.close();
    if(!matchFound) {
        agentsOnCall.add(new Call(agentChanname, callerChanname, queueName,
            linkedid));
    }
}
rs1.close();
```

Výpis 2: Získávání informací o probíhajících hovorech

Princip metody, který je uveden ve výpisu č.2 je následující

1. Provedení SQL dotazu, který vrací hodnoty channame, appdata, peer a linkedid z tabulky CEL, při typu události BRIDGE_ENTER (zahájení přímého spojení) a použití aplikace Queue za posledních 8 hodin.

2. Získané hodnoty jsou poté reprezentovány následovně
    - peer - název kanálu agenta v proměnné agentChanname
    - channame - název kanálu volajícího v proměnné callerChanname
    - appdata - název fronty v proměnné queueName
    - linkedid - ID celé relace v proměnné linkedid
3. Následuje SQL dotaz žádající hodnotu linkedid při typu události BRIDGE_EXIT (ukončení přímého spojení) a shodné linkedid hodnoty jako při dotazu předešlém.
4. Pokud se shoda nenašla, jedná o stále aktivní hovor a uvedené hodnoty v bodu č.2 jsou použity k vytvoření nového objektu v seznamu o probíhajících hovorech.

7.2.2 Metoda pro získání nejčastěji volané fronty

K získání nejčastěji volané fronty za posledních 24 hodin jsem implementoval metodu getMostCalledQueue, která vrací hodnotu typu String s názvem fronty. Ukázka metody je uvedena ve výpisu č.3.

```
query = "SELECT appdata FROM cel "
    + "WHERE eventtype LIKE 'BRIDGE_ENTER' "
    + "AND appname LIKE 'queue' "
    + "AND appdata NOT LIKE '' "
    + "AND (eventtime > DATE_SUB(now(), INTERVAL 24 HOUR)) "
    + "GROUP BY appdata "
    + "ORDER BY COUNT(*) DESC "
    + "LIMIT 1;";

ResultSet rs = sh.getResultSet(query);
while(rs.next()) {
    String queueName = rs.getString("appdata");
    mostCalledQueue = queueName
}
```

Výpis 3: Získání nejčastěji volané fronty

Postup k získání nejvíce volané fronty pomocí záznamů CEL je následující

1. Provedení SQL dotazu nad tabulkou CEL, který vrací hodnotu appdata (název fronty), která se vyskytla nejvíce v záznamech při typu události BRIDGE_ENTER (navázaní přímého spojení) a aplikaci Queue v posledních 24 hodinách.
2. Získaná hodnota je poté uložena do proměnné mostCalledQueue, která je také návratová hodnota metody.

7.2.3 Metoda pro získání počtu čekajících uživatelů ve frontě

Pro zjištění, kolik volajících aktuálně čeká ve frontě na přepojení k agentovi jsem implementoval metodu **getCallersInQueue**. Zdrojový kód metody je uveden v příloze C.2. Tato funkce vrací kolekci typu ArrayList, obsahující hodnoty typu String s názvy kanálů volajících, kteří čekají ve frontě na spojení s agentem.

```
query = "SELECT channame, linkedid FROM cel "
    + "WHERE eventtype LIKE 'APP_START' "
    + "AND appname LIKE 'Queue' "
    + "AND (eventtime > DATE_SUB(now(), INTERVAL 8 HOUR)) "
    + "ORDER BY id DESC;";

ResultSet rs1 = sh.getResultSet(query);
while(rs1.next()) {
    callerChanname = rs1.getString("channame").substring(4, 8);;
    linkedid = rs1.getString("linkedid");

    query = "SELECT DISTINCT linkedid FROM cel "
        + "WHERE (eventtype LIKE 'BRIDGE_ENTER' "
        + "OR eventtype LIKE 'LINKEDID_END') "
        + "AND (eventtime > DATE_SUB(now(), INTERVAL 8 HOUR));";

    ResultSet rs2 = sh.getResultSet(query);
    boolean matchFound = false;

    while(rs2.next()) {
        if(rs2.getString("linkedid").equals(linkedid)) {
            matchFound = true;
        }
    }
    rs2.close();
    if(!matchFound) {
        callersInQueue.add(callerChanname);
    }
}
rs1.close();
```

Výpis 4: Zjištění počet čekajících ve frontě

Na uvedeném výpisu zdrojového kódu č.4 je uveden postup pro zjišťování počtu volajících ve frontě. Postup je následující

1. Provedení SQL dotazu, vracející hodnoty channame (kanál volajícího) a linkedid (ID relace) z tabulky CEL, s typem události APP_START (start sledované aplikace) v posledních 8 hodinách, kde započala aplikace Queue.

2. Následuje uložení hodnoty channame do proměnné callerChanname a hodnoty linkedid do proměnné linkedid.

3. V následujícím kroku je proveden druhý SQL dotaz, který vrací unikátní výpis hodnot linkedid, kde je typ události BRIDGE_ENTER nebo LINKEDID_END, což uvádí, zda bylo navázáno spojení s agentem nebo, zda byla relace již ukončena.

4. Pokud je hodnota uložená v proměnné linkedid z prvního SQL dotazu nenalezena v hodnotách vrácenými pomocí druhého SQL dotazu, tak se jedná o relaci, ve které volající stále čeká ve frontě na přepojení. Následně je hodnota proměnná callerChanname přidána do seznamu callersInQueue.

7.2.4 Metoda pro získání celkového počtu dokončených hovorů

Metoda **getCompletedCalls** slouží ke zjištění počtu dokončených hovorů a příslušných front, které proběhly za posledních 24 hodin. Kompletní zdrojový kód je k dispozici v příloze C.3. Metodu jsem implementoval obdobným způsobem jako metodu getAgentsOnCall uvedenou v kapitole 7.2.1. Návratová hodnota funkce je kolekce typu ArrayList s objekty, které obsahují informace o hodnotách linkedid a appdata, které uvádí ID relací, pro které byly hovory již ukončeny a názvy volaných front.

```
query = "SELECT appdata, linkedid FROM cel "
    + "WHERE eventtype LIKE 'BRIDGE_ENTER' "
    + "AND appname LIKE 'Queue' "
    + "AND (eventtime > DATE_SUB(now(), INTERVAL 24 HOUR)) "
    + "ORDER BY id DESC;";

ResultSet rs1 = sh.getResultSet(query);
while(rs1.next()) {
    queueName = rs1.getString("appdata");
    linkedid = rs1.getString("linkedid");
```

```
        query = "SELECT linkedid FROM cel "
            + "WHERE eventtype LIKE 'BRIDGE_EXIT' "
            + "AND linkedid LIKE '" + linkedid + "' "
            + "AND(eventtime > DATE_SUB(now(), INTERVAL 24 HOUR)) "
            + "ORDER BY id DESC;";

        ResultSet rs2 = sh.getResultSet(query);
        while(rs2.next()) {
            if(rs2.getString("linkedid").equals(linkedid)) {
                completedCalls.add(new Call(null, null, queueName, linkedid);
            }
        }
        rs2.close();
    }
    rs1.close();
```

Výpis 5: Celkový počet dokočených hovorů

Postup pro získání hodnot názvu fronty a ID relace pro ukončené hovory, který je zobrazen ve výpisu č.5 je následující

1. Provedení SQL dotazu, který vrací hodnoty linkedid a appdata z tabulky CEL s typem události BRIDGE_ENTER (zahájení přímého spojení), aplikací Queue a pouze v posledních 24 hodinách.

2. Poté následuje uložení hodnot linkedid a appdata do příslušných proměnných.

3. Následně se provede druhý SQL dotaz vracející pouze hodnotu linkedid, ale nyní pro typ události BRIDGE_EXIT (ukončení přímého spojení) a pro shodnou hodnotu linkedid s uloženou proměnnou.

4. Pokud se tyto dvě hodnoty shodují, znamená to, že hovor již proběhl a proměnné queueName a linkedid jsou přidány do kolekce obsahující ID relace proběhlých hovorů s příslušnými frontami.

7.2.5 Metoda pro získání aktivních agentů

Metoda **getActiveAgents** slouží k získání aktivních agentů a front, ke kterým jsou agenti přihlášení. Kompletní zdrojový kód této metody je uveden v příloze C.4. Návratovou hodnotou metody je kolekce ArrayList s objekty obsahující název kanálu aktivního agenta a frontu, ke které je přihlášen.

```
query = "SELECT MAX(eventtime) AS eventtime, "
    + "cid_num, appdata FROM cel "
    + "WHERE eventtype LIKE 'APP_START' "
    + "AND appname LIKE 'AddQueueMember' "
    + "GROUP BY cid_num, appname, appdata;";

ResultSet rs1 = sh.getResultSet(query);
while(rs1.next()) {
    agentNumber = rs1.getString("cid_num");
    queueName = rs1.getString("appdata");
    eventtime = rs1.getString("eventtime");

    query = "SELECT cid_num FROM cel "
        + "WHERE eventtype LIKE 'APP_START' "
        + "AND appname LIKE 'RemoveQueueMember' "
        + "AND cid_num LIKE '" + agentNumber + "' "
        + "AND appdata LIKE '" + queueName + "' "
        + "AND eventtime > '" + eventtime + "' "
        + "ORDER BY id DESC;";

    ResultSet rs2 = sh.getResultSet(query);
    boolean matchFound = false;
    while(rs2.next()) {
        if(rs2.getString("cid_num").equals(agentNumber)) {
            matchFound = true;
        }
    }
```

```
        rs2.close();
        if(!matchFound) {
            activeAgents.add(new Call(agentNumber, null, queueName, null));
        }
    }
    rs1.close();
```

Výpis 6: Získání aktivních agentů

Postup k získání informací o aktivních agentech je uveden ve výpisu č.6. Samotný postup je následující

1. Provedení SQL dotazu, vracející hodnoty nejaktuálnějšího času přihlášení agenta cid_num (číslo agenta) a appdata (název fronty) z tabulky CEL, při startu aplikace (APP_START) pro přidání agenta do fronty (AddQueueMember).
2. Následně jsou získané hodnoty uloženy do příslušných proměnných.
3. Provedení druhého SQL dotazu, vracející cid_num (číslo agenta), při aplikaci pro odebrání agenta z fronty (RemoveQueueMember), hodnotách názvu fronty a kanálu agenta shodnými s výše uloženými proměnnými a časovým údajem aktuálnějším než v uložené proměnné. Tímto se zjišťuje, zda se agent odhlásil od fronty ke které se v čase před touto událostí přihlásil.
4. Pokud kanál agenta s příslušnou frontou nebyl vrácen druhým SQL dotazem, tak je přidán do seznamu obsahující aktivní agenty.

7.3 Další třídy využívané webovou aplikací

Následující sekce popisují třídy, které obsahují klíčové metody pro funkčnost aplikace.

7.3.1 Navázaní spojení s databází

K navázání spojení s databází slouží třída DatabaseConnection, uvedena v příloze C.5. Konstruktor vyžaduje název databáze. Při vytvoření instance třídy je navázáno spojení s databází pomocí konektoru JDBC. Datové spojení je reprezentováno proměnnou typu DataSource.

7.3.2 Zpracování SQL dotazů

Ke zpracování SQL dotazů jsem implementoval třídu SqlHelper, která je uvedena v příloze C.6. Konstruktor třídy má za parametr připojení typu Connection, které je získáno metodou getConnection z proměnné typu DataSource.

Metoda getResultSet bere jako parametr samotný SQL dotaz jako typ String. Poté je dotaz zpracován pomocí rozhraní PreparedStatement, které zpracuje SQL dotaz nad databází. Získané informace jsou poté uloženy v proměnné typu ResultSet. Samotná metoda vrací proměnnou typu ResultSet.

7.3.3 Třída s informacemi o probíhajícím hovoru

Tato třída slouží k vytváření instancí, které obsahují informace pro identifikaci probíhajícího hovoru. Mezi tyto informace patří kanál použitý agentem, kanál použitý volajícím, název fronty a ID relace. Třída také obsahuje metody pro získání uvedených informací. Zdrojový kód třídy je uveden v příloze C.7.

7.4 Realizace grafického rozhraní aplikace

Samotné grafické rozhraní (GUI), které slouží uživatelům k monitorování informací získaných ze záznamů CEL jsem realizoval jako HTML dokument, který je poskytnut jako webová stránka pomocí servletu. Servlet je program napsaný v jazyce Java, který je součástí webového serveru a nástrojem pro tvorbu webových aplikací. Funguje na principu zpracování HTTP požadavků a generování HTML stránek.

Pro vzhledovou část webového GUI jsem využil primárně volně dostupný framework Bootstrap, což je nástroj pro tvorbu webových aplikací pomocí HTML, CSS a JavaScriptu.

7.4.1 Zpracování dat do HTML dokumentu

Po přijetí HTTP požadavku uživatele, je požadavek předán do servlet kontejneru. Ten následně namapuje požadavek na příslušný servlet. V servlet metodě init, která je volána jako první, je navázáno spojení s databází pomocí třídy DatabaseConnection. Metoda init je uvedena ve výpisu č.7.

```
private final String databaseName = "asterisk";
private DatabaseConnection dbc;

@Override
public void init( ServletConfig config ) {
    dbc = new DatabaseConnection(databaseName);
}
```

Výpis 7: Servlet metoda init

Následuje metoda doGet, která již zasílá data na výstup. V mém případě vytváří samotnou webovou stránku. Metoda doGet obsahuje nastavení typu výstupu na HTML a data na samotném výstupu. Metoda je uvedena ve výpisu č.8.

```
@Override
public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

    response.setContentType("text/html");

    PrintWriter out = response.getWriter();

    displayDashboard(out);
}
```

Výpis 8: Servlet metoda doGet

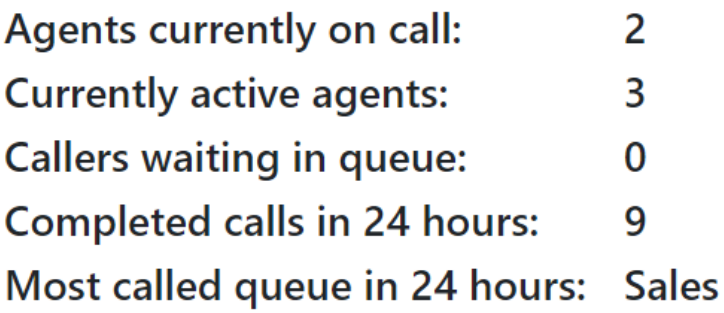
Metodu displayDashboard jsem implementoval pro zpracování dat získaných pomocí metod uvedených v kapitole č.7.2. Metoda vrací tyto data zpracované pro zobrazení ve HTML formátu.

Zobrazení statistik jsem realizoval pomocí tabulky obsahující dva sloupce, a to pro název zobrazované informace a data získané z metod pro práci s databází. Ukázka je zobrazena ve výpisu č.9.

```
...
content += "<table><tbody><tr>";
content += "<td><h3>Agents currently on call:</h3></td>";
content += "<td><h3>"+ dashboard.getAgentsOnCall().size() +"</td>";
content += "</tr><tr>";
content += "<td><h3>Callers waiting in queue:</h3></td>";
content += "<td><h3>"+ dashboard.getCallersInQueue().size() +"</h3></td>";
content += "</tr><tr>";
content += "<td><h3>Completed calls in 24 hours:</td>";
content += "<td><h3>"+ dashboard.getCompletedCalls().size() +"</h3></td>";
content += "</tr><tr>";
content += "<td><h3>Most called queue in 24 hours:</h3></td>";
content += "<td><h3>"+ dashboard.getMostCalledQueue() +"</h3></td>";
content += "</tr></tbody></table>";
out.print(content);
```

Výpis 9: Výpis statistik monitorování v HTML formátu

Při zobrazení výše uvedeného HTML výstupu pomocí webového prohlížeče, budou zobrazeny informace v následujícím formátu (obrázk č.15).



Obrázek 15: Zobrazení statistik monitorování v prohlížeči

Zobrazení seznamu s informacemi o aktivních agentech a probíhajících hovorech jsem implementoval pomocí HTML tabulky, do které jsou přidávány záznamy (řádky) s příslušnými informacemi za každého aktivního agenta nebo probíhající hovor. Seznam s hovory se zobrazí pouze tehdy, pokud probíhá minimálně jeden hovor. Ukázka implementace seznamu s informacemi o probíhajících hovorech je uvedena ve výpisu č.10. Implementace seznamu aktivních agentů je vyřešena obdobným způsobem (ukázka implementace v příloze C.8).

```
if (dashboard.getAgentsOnCall().size() > 0) {
    ...
    for (int i = 0; i < dashboard.getAgentsOnCall().size(); i++) {
        content += "<tr>";
        content += "<td><h4 align=\"center\">"
            + dashboard.getAgentsOnCall().get(i).getAgentChanname() +"</h4></td>
                ";
        content += "<td><img src=\"images/green_arrow_1.png\" style=\"height:30
            px;\"></td>";
        content += "<td><h4 align=\"center\">"
            + dashboard.getAgentsOnCall().get(i).getCallerChanname() + "</h4></
                td>";
        content += "<td><h4 align=\"center\">" +
            + dashboard.getAgentsOnCall().get(i).getQueueName() + "</h4></td>";
        content += "</tr>";
    }
}
out.print(content);
```

Výpis 10: Výpis informací o aktivních hovorech v HTML formátu

Pokud jsou aktivní tři agenti a zrovna probíhají dva hovory, tak ukázka uvedeného HTML výstupu při zobrazení ve webovém prohlížeči bude vypadat následovně (obrázek č.16).

Obrázek 16: Zobrazení informací o aktivních agentech a probíhajících hovorech

Webové rozhraní také obsahuje výsečový graf, který zobrazuje informace o počtu dokončených hovorů ve specifických frontách. Graf jsem realizoval pomocí volně dostupného nástroje Chart.js, který slouží tvorbě HTML grafů.

K získání dat pro graf je nejprve použit cyklus ke spočítání výskytu specifických front v seznamu s dokončenými hovory. Poté jsou hodnoty výskytu a názvy front předány do samotného grafu. Tento postup je uveden ve výpisu č.11.

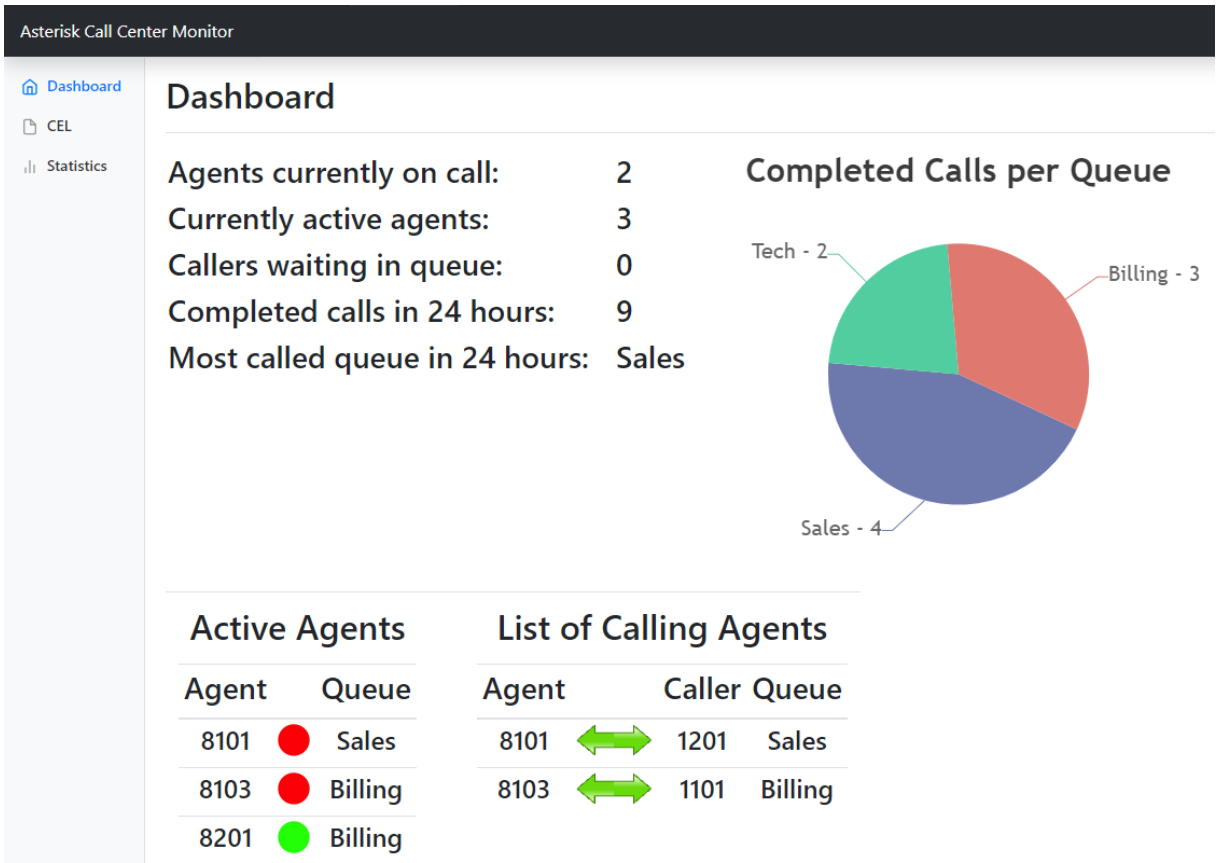
```
...
for (int i = 0; i < completedCalls.size(); i++) {
    if(completedCalls.get(i).getQueueName().equals("Sales"))
        salesCounter++;
    else if(completedCalls.get(i).getQueueName().equals("Tech"))
        techCounter++;
    else if(completedCalls.get(i).getQueueName().equals("Billing"))
        billingCounter++;
    }

content += "dataPoints: ["

if(salesCounter > 0)
    content += "{ y:" + salesCounter + ", label: \"Sales\" },\n";
if(techCounter > 0)
    content += "{ y:" + techCounter + ", label: \"Tech\" },\n";
if(billingCounter > 0)
    content += "{ y:" + billingCounter + ", label: \"Billing\" },\n";
...
```

Výpis 11: Předání informací o volaných frontách do grafu

Dynamickou aktualizaci zobrazených monitorovacích informací jsem realizoval HTML funkcí, která obnovuje obsah webové stránky na straně klienta každé tři vteřiny. Další možností, kterou by se dalo využít pro dynamické načítání dat z databáze by mohla být technologie AJAX (Asynchronous JavaScript and XML). Využití AJAXu by pro navržený systém bylo více praktické, jelikož dokáže měnit obsah bez kompletního znovunačtení webové stránky.

7.5 Výsledné webové rozhraní

Na obrázku č.17 je znázorněno výsledné webové rozhraní aplikace pro monitorování. Aplikace slouží k získání základních informací jako seznam aktivních agentů, informace o probíhajících hovorech a statistiky o dokončených hovorech v systému Asterisk apod.



Obrázek 17: Ukázka výsledného webového rozhraní

Hlavní stranu monitorovacího rozhraní jsem rozdělil do čtyř části, kterými jsou

- Statistiky
- Seznam aktivních agentů
- Seznam volajících agentů
- Dokončené hovory v určitých frontách a jejich poměr

Část obsahující statistiky je velmi flexibilní, jelikož je rozšířitelná o jakékoliv další dostupné informace, které se dají získat ze záznamů CEL. Může se jednat např. o nejdelší hovor, nejvíce aktivního agenta, celkový provolaný čas za posledních 24 hodin apod.

Seznam s aktivními agenty zobrazuje agenty, kteří jsou aktuálně přihlášeni a jejich příslušnou frontu, za kterou jsou zodpovědní. Informace, o kterou by údaje o aktivních agentech mohly být rozšířeny je např. celkový čas, po který jsou agenti přihlášeni.

Seznam obsahující aktuálně volající agenty zobrazuje základní informace o probíhajícím hovoru jako číslo kanálu agenta, číslo kanálu volajícího a název fronty. Tento seznam je možno rozšířit o informace jako aktuální délka hovoru nebo počet dokončených hovorů agenta.

Poslední část obsahuje výsečový graf, který znázorňuje počet dokončených hovorů v určitých frontách a poměr mezi nimi. Dále by bylo monitorovací rozhraní možno rozšířit např. o spojnicový graf, uvádějící počet dokončených hovorů na každou hodinu za poslední den.

Závěr

Cílem mé bakalářské práce byla realizace odlehčeného webového rozhraní pro monitoring, které by mohlo být využito pro správu menších call center.

První kapitola teoretické části práce se stručně věnovala technologii VoIP a protokolům, především signalizačnímu protokolu SIP a jeho funkcím. V další kapitole byl rozebrán systém Asterisk a jeho funkce. Zaměření této kapitoly bylo především na systém pro vytváření záznamů CEL, a poté také funkce, které jsou využívány při realizaci menšího call centra. Mezi tyto funkce patří např. interaktivní hlasový průvodce a fronty. Poslední část kapitoly zabývající se systémem Asterisk byla zaměřena na jeho integraci s databázovými systémy. Poslední kapitola teoretické části stručně popsala pojem webový server, jeho komunikaci s klienty, HTTP protokol a realizaci webové aplikace.

Praktická část práce byla rozdělena do čtyř kapitol. První z nich se zabývala rozvržením realizace praktické části, znázorněním komunikace systémů mezi sebou a popsala použitý software. Ve druhé kapitole byl sepsán postup pro instalaci a konfiguraci systému Asterisk pro funkci jako menší call centrum. Kapitola obsahuje také aktivaci funkce CEL a její konfiguraci. Třetí kapitola řešila realizaci propojení výstupu funkce CEL s MySQL databází pomocí konektoru ODBC. Poslední kapitola praktické části řešila realizaci webové Java aplikace pro monitorování. Hlavní důraz v této kapitole byl kladen na metody použité pro získávání potřebných dat z databáze pro monitoring. Závěrem bylo předvedení výsledného grafického rozhraní webové aplikace.

Monitorovací aplikace byla navržena s minimální závislostí na ostatních systémech. Aplikace je závislá pouze na poskytnutých CEL záznamech. Na konfiguraci Asterisku, jako názvy kanálů, nastavení front apod. nezáleží. Webová aplikace je také schopna spolupracovat s ostatními RDBMS namísto MySQL, jako jsou PostgreSQL nebo Microsoft SQL, pokud bude zachován formát tabulky pro záznamy CEL a pozměněn příslušný JDBC ovladač.

Výslednou aplikaci je možno využít v menších call centrech k monitorování dostupných agentů, probíhajících hovorů, počet dokončených hovorů apod. Dále by bylo možno aplikaci také využít v organizacích, které využívají ke komunikaci technologii VoIP na platformě Asterisk PBX. Využití může zahrnovat monitorování hovorů mezi členy organizace nebo počtu uskutečněných hovorů. Při provedení určitých změn, by aplikace mohla být také využita jako podklad k monitorování informací v úplně jiných prostředích než v telefonii. Mezi tato prostředí by mohla patřit např. správa e-shopu nebo monitorování webového zákaznického servisu. Hlavním důvodem, proč by měla být navržená aplikace zvolena je minimální závislost na zvoleném databázovém systému a na konfiguraci systému Asterisk a rovněž velké možnosti rozšířitelnosti webového rozhraní o požadované údaje k monitorování.

Odkazy

1. PCWORLD.CZ. *VoIP - úvod do problematiky volání po internetu: Úvod do problematiky VoIP* [online]. 2009 [cit. 2019-04-12]. Dostupné z: http://pcworld.cz/internet/voip-uvod-do-problematiky-volani-po-internetu-8120.
2. JOYCE ČR, S.R.O. *Vše o VoIP - rady, tipy: 1. díl - Základy VoIP* [online]. 2019 [cit. 2019-04-12]. Dostupné z: https://www.joyce.cz/cz/voip-reseni/vse-o-voip-rady-tipy/1-dil-zaklady-voip/.
3. INFORMATION SCIENCES INSTITUTE UNIVERSITY. *INTERNET PROTOCOL* [online]. 2019 [cit. 1981-09]. Dostupné z: https://tools.ietf.org/html/rfc791.
4. VOZŇÁK, M.; ROZHON, J. *Spojovací systémy: SIP - Session Initiation Protocol* [online]. 2018 [cit. 2019-04-13]. Dostupné z: https://drive.google.com/file/d/1QTAR1w-Hz6AqJ3DItbAuDzw0Xd3dkEZ_/view.
5. VOZŇÁK, M. *SIP/SDP protocols – elementrs, requests and responses, headers: Session Initiation Protocol* [online]. 2018 [cit. 2019-04-13]. Dostupné z: https://lms.vsb.cz/mod/resource/view.php?id=386957.
6. VOIPTHINK. *SIP Protocol: SIP Architecture* [online]. 2019 [cit. 2019-04-13]. Dostupné z: http://www.en.voipforo.com/SIP/SIP_architecture.php.
7. BERNERS-LEE, T. *Uniform Resource Identifier (URI): Generic Syntax* [online]. 2018 [cit. 2019-04-13]. Dostupné z: https://tools.ietf.org/html/rfc3986.
8. NETWORK WORLD. *What is SIP?* [online] [cit. 2019-04-12]. Dostupné z: https://www.networkworld.com/article/2332980/lan-wan-what-is-sip.html.
9. DIGIUM, INC. *Asterisk Administrator Guide* [online]. 2018 [cit. 2019-04-12]. Dostupné z: https://wiki.asterisk.org/wiki/download/attachments/19005471/asterisk-admin-guide.pdf.
10. DIGIUM, INC. *Asterisk Architecture, The Big Picture: An Asterisk System* [online]. 2014 [cit. 2019-04-13]. Dostupné z: http://www.abclinuxu.cz/clanky/site/asterisk-voip-ustredna-2-konfigurace.
11. J. G. VAN BOSSE, F. U. Devetak. *Signaling in Telecommunication Networks.* 2007. ISBN 978-0-471-66288-4.
12. VALOUŠEK, O. *Asterisk: VoIP ústředna - 2 (konfigurace): 2. extensions.conf* [online]. 2006 [cit. 2019-04-13]. Dostupné z: http://www.abclinuxu.cz/clanky/site/asterisk-voip-ustredna-2-konfigurace.
13. DIGIUM, INC. *Creating SIP Accounts* [online]. 2014 [cit. 2019-04-12]. Dostupné z: https://wiki.asterisk.org/wiki/display/AST/Creating+SIP+Accounts.

14. VOIP-INFO.ORG LLC. *Asterisk config sip.conf* [online]. 2018 [cit. 2019-04-13]. Dostupné z: `https://www.voip-info.org/asterisk-config-sipconf/`.

15. BRYANT, R.; MADSEN, L.; MEGGELEN, J. V. *Asterisk^{TM}: The Definitive Guide: The Future of Telephony Is Now.* 4th ed. Ed. by LOUKIDES, M.; JEPSON, N. United States of America: O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2013. ISBN 978-1-449-33242-6.

16. VOIP-INFO.ORG LLC. *Asterisk call queues* [online]. 2018 [cit. 2019-04-12]. Dostupné z: `https://www.voip-info.org/asterisk-call-queues/`.

17. DIGIUM, INC. *IVR System: What Is An IVR System?* [online]. 2019 [cit. 2019-04-13]. Dostupné z: `https://www.asterisk.org/get-started/applications/ivr`.

18. PYTHON DOCUMENTATION. *Python's strftime directives* [online]. 2015 [cit. 2019-04-18]. Dostupné z: `http://strftime.org/`.

19. EASYSOFT LIMITED. *Linux/UNIX ODBC: What is ODBC?* [online]. 2019 [cit. 2019-04-14]. Dostupné z: `https://www.easysoft.com/developer/interfaces/odbc/linux.html#what_is_odbc`.

20. MOZILLA CORPORATION. *What is a web server?* [online]. 2019 [cit. 2019-04-14]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server`.

21. FIELDING, R.; GETTYS, J.; MOGUL, J.; FRYSTYK, H.; MASINTER, L.; LEACH, P.; BERNERS-LEE, T. *Hypertext Transfer Protocol – HTTP/1.1* [online]. 1999 [cit. 2019-04-18]. Dostupné z: `https://tools.ietf.org/html/rfc2616`.

22. ORACLE CORPORATION. *JDBC* [online]. 2002 [cit. 2019-04-18]. Dostupné z: `https://docs.oracle.com/javase/1.5.0/docs/guide/jdbc/`.

23. CHAFFEE, A. *What is a web application (or "webapp")?* [online]. 2012 [cit. 2019-04-14]. Dostupné z: `http://www.jguru.com/faq/view.jsp?EID=129328`.

A Konfigurační soubory Asterisku

A.1 Výpis konfigurace sip.conf

```
[general]
context=public
allowoverlap=no
udpbindaddr=0.0.0.0
tcpenable=no
tcpbindaddr=0.0.0.0
transport=udp
srvlookup=yes
allowguest=no
externip=81.200.53.38
externaddr=81.200.53.38
localnet=192.168.0.0/255.255.0.0

[users_a](!)
type=friend
qualify=yes
nat=yes
host=dynamic
canreinvite=no
context=user_group_a

[1101](users_a)
secret=123
mailbox=1101@voicemail_user_group_a

[1102](users_a)
secret=123
mailbox=1102@voicemail_user_group_a

[1103](users_a)
secret=123
mailbox=1103@voicemail_user_group_a

[1104](users_a)
secret=123
```

```
mailbox=1104@voicemail_user_group_a

[users_b](!)
type=friend
qualify=yes
nat=yes
host=dynamic
canreinvite=no
context=user_group_b

[1201](users_b)
secret=123
mailbox=1201@voicemail_user_group_b

[1202](users_a)
secret=123
mailbox=1202@voicemail_user_group_b

[1203](users_a)
secret=123
mailbox=1203@voicemail_user_group_b

[1204](users_a)
secret=123
mailbox=1204@voicemail_user_group_b

[agents_a](!)
type=friend
qualify=yes
nat=yes
host=dynamic
canreinvite=no
context=agent_group_a

[8101](agents_a)
secret=123

[8102](agents_a)
secret=123
```

```
[8103](agents_a)
secret=123

[8104](agents_a)
secret=123

[agents_b](!)
type=friend
qualify=yes
nat=yes
host=dynamic
canreinvite=no
context=agent_group_b

[8201](agents_b)
secret=123

[8202](agents_b)
secret=123

[8203](agents_b)
secret=123

[8204](agents_b)
secret=123
```

A.2 Výpis konfigurace queues.conf

```
[general]
persistentmembers=yes
musicclass=default
strategy=ringall
timeout=15
wrapuptime=15
autofill=no
maxlen=0
joinempty=yes
leavewhenempty=no
reportholdtime=no
keepstats=no

[sales]
strategy = rrmemory

[tech]
strategy = leastrecent

[billing]
strategy = fewestcalls
```

A.3 Výpis konfigurace extensions.conf

```
[general]
static=yes
writeprotect=no
clearglobalvars=no

[globals]
CONSOLE=Console/dsp
IAXINFO=guest
TRUNK=DAHDI/G2
TRUNKMSD=1

[user_group_a]
exten => 1101,1,Dial(SIP/1101,30,m)
exten => 1101,n,Voicemail(1101@voicemail_user_group_a)
exten => 1101,n,Hangup

exten => 1102,1,Dial(SIP/1102,30,m)
exten => 1102,n,Voicemail(1102@voicemail_user_group_a)
exten => 1102,n,Hangup

exten => 1103,1,Dial(SIP/1103,30,m)
exten => 1103,n,Voicemail(1103@voicemail_user_group_a)
exten => 1103,n,Hangup

exten => 1104,1,Dial(SIP/1104,30,m)
exten => 1104,n,Voicemail(1104@voicemail_user_group_a)
exten => 1104,n,Hangup

exten => 1199,1,VoicemailMain(@voicemail_user_group_a)

exten => 911,1,Goto(ivr_user_menu,s,1)

[user_group_b]
exten => 1201,1,Dial(SIP/1201,30,m)
exten => 1201,n,Voicemail(1201@voicemail_user_group_b)
exten => 1201,n,Hangup
```

```
exten => 1202,1,Dial(SIP/1202,30,m)
exten => 1202,n,Voicemail(1202@voicemail_user_group_b)
exten => 1202,n,Hangup

exten => 1203,1,Dial(SIP/1203,30,m)
exten => 1203,n,Voicemail(1203@voicemail_user_group_b)
exten => 1203,n,Hangup

exten => 1204,1,Dial(SIP/1204,30,m)
exten => 1204,n,Voicemail(1204@voicemail_user_group_b)
exten => 1204,n,Hangup

exten => 1299,1,VoicemailMain(@voicemail_user_group_b)

exten => 911,1,Goto(ivr_user_menu,s,1)

[agent_group_a]
exten => 911,1,Goto(ivr_agent_menu,s,1)

[agent_group_b]
exten => 911,1,Goto(ivr_agent_menu,s,1)

[ivr_user_menu]
exten => s,1,Answer
exten => s,2,Background(/var/lib/asterisk/sounds/ivr_promt_user)
exten => s,n,WaitExten

exten => 1,1,Playback(/var/lib/asterisk/sounds/sales_message)
exten => 1,n,NoOp(waiting)
exten => 1,n,Background(queue-callswaiting)
exten => 1,n,Queue(sales)

exten => 2,1,Playback(/var/lib/asterisk/sounds/tech_message)
exten => 2,n,NoOp(waiting)
exten => 2,n,Background(queue-callswaiting)
exten => 2,n,Queue(tech)

exten => 3,1,Playback(/var/lib/asterisk/sounds/billing_message)
exten => 3,n,NoOp(waiting)
```

```
exten => 3,n,Background(queue-callswaiting)
exten => 3,n,Queue(billing)

exten => 0,1,MusicOnHold(default)
exten => 0,n,Goto(ivr_user_menu,s,2)

exten => #,1,Playback(vm-goodbye)
exten => #,n,Hangup

exten => t,1,Goto(#,1)
exten => i,1,Playback(invalid)

[ivr_agent_menu]
exten => s,1,Answer
exten => s,2,Background(agent-user)
exten => s,n,WaitExten

exten => 1,1,Playback(agent-loginok)
exten => 1,n,Goto(ivr_agent_login,s,1)

exten => 2,1,Playback(agent-loggedoff)
exten => 2,n,Goto(ivr_agent_logout,s,1)

exten => 0,1,MusicOnHold(default)
exten => 0,n,Goto(ivr_main,911,2)

exten => #,1,Playback(vm-goodbye)
exten => #,n,Hangup

exten => t,1,Goto(#,1)
exten => i,1,Playback(invalid)

[ivr_agent_login]
exten => s,1,Background(/var/lib/asterisk/sounds/ivr_promt_user)
exten => s,n,WaitExten

exten => 1,1,AddQueueMember(sales)
exten => 1,n,Playback(/var/lib/asterisk/sounds/sales_message)
exten => 1,n,Playback(agent-loginok)
```

```
exten => 1,n,Goto(#,1)

exten => 2,1,AddQueueMember(tech)
exten => 2,n,Playback(/var/lib/asterisk/sounds/tech_message)
exten => 2,n,Playback(agent-loginok)
exten => 2,n,Goto(#,1)

exten => 3,1,AddQueueMember(billing)
exten => 3,n,Playback(/var/lib/asterisk/sounds/billing_message)
exten => 3,n,Playback(agent-loginok)
exten => 3,n,Goto(#,1)

exten => #,1,Playback(vm-goodbye)
exten => #,n,Hangup

exten => t,1,Goto(#,1)
exten => i,1,Playback(invalid)

[ivr_agent_logout]
exten => s,1,Background(/var/lib/asterisk/sounds/ivr_promt_user)
exten => s,n,WaitExten

exten => 1,1,RemoveQueueMember(sales)
exten => 1,n,Playback(/var/lib/asterisk/sounds/sales_message)
exten => 1,n,Playback(agent-loggedoff)
exten => t,1,Goto(#,1)

exten => 2,1,RemoveQueueMember(tech)
exten => 2,n,Playback(/var/lib/asterisk/sounds/tech_message)
exten => 2,n,Playback(agent-loggedoff)
exten => t,1,Goto(#,1)

exten => 3,1,RemoveQueueMember(billing)
exten => 3,n,Playback(/var/lib/asterisk/sounds/billing_message)
exten => 3,n,Playback(agent-loggedoff)
exten => t,1,Goto(#,1)

exten => #,1,Playback(vm-goodbye)
exten => #,n,Hangup
```

```
exten => t,1,Goto(#,1)
exten => i,1,Playback(invalid)
```

B Konfigurace MySQL serveru

B.1 Vytvoření tabulky pro výstup záznamů CEL

```
CREATE TABLE cel (
id BIGINT(20) UNSIGNED NOT NULL AUTO_INCREMENT,
eventtype VARCHAR(30) COLLATE utf8_unicode_ci NOT NULL,
eventtime TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
    CURRENT_TIMESTAMP,
userdeftype VARCHAR(255) COLLATE utf8_unicode_ci NOT NULL,
cid_name VARCHAR(80) COLLATE utf8_unicode_ci NOT NULL,
cid_num VARCHAR(80) COLLATE utf8_unicode_ci NOT NULL,
cid_ani VARCHAR(80) COLLATE utf8_unicode_ci NOT NULL,
cid_rdnis VARCHAR(80) COLLATE utf8_unicode_ci NOT NULL,
cid_dnid VARCHAR(80) COLLATE utf8_unicode_ci NOT NULL,
exten VARCHAR(80) COLLATE utf8_unicode_ci NOT NULL,
context VARCHAR(80) COLLATE utf8_unicode_ci NOT NULL,
channame VARCHAR(80) COLLATE utf8_unicode_ci NOT NULL,
appname VARCHAR(80) COLLATE utf8_unicode_ci NOT NULL,
appdata VARCHAR(80) COLLATE utf8_unicode_ci NOT NULL,
amaflags INT(11) NOT NULL,
accountcode VARCHAR(20) COLLATE utf8_unicode_ci NOT NULL,
peeraccount VARCHAR(20) COLLATE utf8_unicode_ci NOT NULL,
uniqueid VARCHAR(150) COLLATE utf8_unicode_ci NOT NULL,
linkedid VARCHAR(150) COLLATE utf8_unicode_ci NOT NULL,
userfield VARCHAR(255) COLLATE utf8_unicode_ci NOT NULL,
peer VARCHAR(80) COLLATE utf8_unicode_ci NOT NULL,
UNIQUE KEY id (id)
) ENGINE=INNODB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

Výpis 12: Příkaz pro vytvoření tabulky cel v MySQL

C Výpisy zdrojového kódu webové aplikace

C.1 Zdrojový kód metody pro získání informací o probíhajícím hovoru

```
...
public ArrayList<Call> getAgentsOnCall() {
    try {
        agentsOnCall = new ArrayList<>();

        String agentChanname = "";
        String callerChanname = "";
        String queueName = "";
        String linkedid = "";
        String query = "";

        query = "SELECT channame, appdata, peer, linkedid FROM cel "
            + "WHERE eventtype LIKE 'BRIDGE_ENTER' "
            + "AND channame LIKE 'SIP/%' "
            + "AND appname LIKE 'Queue' "
            + "AND peer LIKE 'SIP/%' "
            + "AND (eventtime > DATE_SUB(now(), INTERVAL 8 HOUR)) "
            + "ORDER BY id DESC;";

        ResultSet rs1 = sh.getResultSet(query);
        while(rs1.next()) {
            agentChanname = rs1.getString("peer").substring(4, 8);
            callerChanname = rs1.getString("channame").substring(4, 8);
            queueName = rs1.getString("appdata").substring(0, 1).toUpperCase()
                + rs1.getString("appdata").substring(1);
            linkedid = rs1.getString("linkedid");

            query = "SELECT linkedid FROM cel "
                + "WHERE eventtype LIKE 'BRIDGE_EXIT' "
                + "AND linkedid LIKE '" + linkedid + "' "
                + "AND(eventtime > DATE_SUB(now(), INTERVAL 8 HOUR)) "
                + "ORDER BY id DESC;";

            ResultSet rs2 = sh.getResultSet(query);
            boolean matchFound = false;
```

```
            while(rs2.next()) {
                if(rs2.getString("linkedid").equals(linkedid)) {
                    matchFound = true;
                }
            }
            rs2.close();
            if(!matchFound) {
                agentsOnCall.add(new Call(agentChanname, callerChanname,
                    queueName, linkedid));
            }
        }
        rs1.close();
    }
    catch(SQLException e) {
        System.out.println(e);
    }
    finally {
        try {
            if(sh.getPreparedStatement() != null) {
                sh.getPreparedStatement().close();
            }
        }
        catch(SQLException e) {
            System.out.println(e);
        }
    }
    return agentsOnCall;
}
...
```

Výpis 13: Metoda getAgentsOnCall

C.2 Zdrojový kód metody pro zjištění počtu čekajících uživatelů ve frontě

```
...
public ArrayList<String> getCallersInQueue() {
    try {
        callersInQueue = new ArrayList<>();

        String callerChanname = "";
        String linkedid = "";
        String query = "";

        query = "SELECT channame, linkedid FROM cel "
            + "WHERE eventtype LIKE 'CHAN_START' "
            + "AND exten LIKE '911' "
            + "AND (eventtime > DATE_SUB(now(), INTERVAL 8 HOUR)) "
            + "ORDER BY id DESC;";

        ResultSet rs1 = sh.getResultSet(query);
        while(rs1.next()) {
            callerChanname = rs1.getString("channame").substring(4, 8);
            linkedid = rs1.getString("linkedid");

            query = "SELECT DISTINCT linkedid FROM cel "
                + "WHERE (eventtype LIKE 'BRIDGE_ENTER' "
                + "OR eventtype LIKE 'LINKEDID_END') "
                + "AND (eventtime > DATE_SUB(now(), INTERVAL 8 HOUR));";

            ResultSet rs2 = sh.getResultSet(query);
            boolean matchFound = false;
            while(rs2.next()) {
                if(rs2.getString("linkedid").equals(linkedid)) {
                    matchFound = true;
                }
            }
            rs2.close();
                if(!matchFound) {
                    callersInQueue.add(caller);
            }
        }
```

```
            rs1.close();
        }
        catch(SQLException e) {
            System.out.println(e);
        }
        finally {
            try {
                if(sh.getPreparedStatement() != null) {
                    sh.getPreparedStatement().close();
                }
            }
            catch(SQLException e) {
                System.out.println(e);
            }
        }
        return callersInQueue;
    }
    ...
```

Výpis 14: Metoda getCallersInQueue

C.3 Zdrojový kód metody pro získání celkového počtu dokončených hovorů

```
...
public ArrayList<Call> getCompletedCalls() {
    try {
        completedCalls = new ArrayList<>();

        String queueName = "";
        String linkedid = "";
        String query = "";

        query = "SELECT appdata, linkedid FROM cel "
            + "WHERE eventtype LIKE 'BRIDGE_ENTER' "
            + "AND channame LIKE 'SIP/%' "
            + "AND appname LIKE 'Queue' "
            + "AND (eventtime > DATE_SUB(now(), INTERVAL 24 HOUR)) "
            + "ORDER BY id DESC;";

        ResultSet rs1 = sh.getResultSet(query);
        while(rs1.next()) {
            queueName = rs1.getString("appdata").substring(0, 1).toUpperCase() +
                rs1.getString("appdata").substring(1);
            linkedid = rs1.getString("linkedid");

            query = "SELECT linkedid FROM cel "
                + "WHERE eventtype LIKE 'BRIDGE_EXIT' "
                + "AND appname LIKE 'Queue' "
                + "AND linkedid LIKE '" + linkedid + "' "
                + "AND(eventtime > DATE_SUB(now(), INTERVAL 24 HOUR)) "
                + "ORDER BY id DESC;";

            ResultSet rs2 = sh.getResultSet(query);
            while(rs2.next()) {
                if(rs2.getString("linkedid").equals(linkedid)) {
                    completedCalls.add(new Call(null, null, queueName, linkedid))
                        ;
                }
            }
            rs2.close();
```

```
        }
        rs1.close();
    }
    catch(SQLException e) {
        System.out.println(e);
    }
    finally {
        try {
            if(sh.getPreparedStatement() != null) {
                sh.getPreparedStatement().close();
            }
        }
        catch(SQLException e) {
                System.out.println(e);
        }
    }
    return completedCalls;
}
...
```

Výpis 15: Metoda getCompletedCalls

C.4 Zdrojová kód metody pro získání aktivních agentů

```
...
public ArrayList<Call> getActiveAgents() {
    try {
        activeAgents = new ArrayList<>();

        String agentNumber = "";
        String queueName = "";
        String eventtime = "";
        String query = "";

        query = "SELECT MAX(eventtime) AS eventtime, cid_num, appdata FROM cel
            "
            + "WHERE eventtype LIKE 'APP_START' "
            + "AND appname LIKE 'AddQueueMember' "
            + "GROUP BY cid_num, appname, appdata;";

        ResultSet rs1 = sh.getResultSet(query);
        while(rs1.next()) {
            agentNumber = rs1.getString("cid_num");
            queueName = rs1.getString("appdata");
            eventtime = rs1.getString("eventtime");

            query = "SELECT cid_num FROM cel "
                + "WHERE eventtype LIKE 'APP_START' "
                + "AND appname LIKE 'RemoveQueueMember' "
                + "AND cid_num LIKE '" + agentNumber + "' "
                + "AND appdata LIKE '" + queueName + "' "
                + "AND eventtime > '" + eventtime + "' "
                + "ORDER BY id DESC;";

            ResultSet rs2 = sh.getResultSet(query);
            boolean matchFound = false;
            while(rs2.next()) {
                if(rs2.getString("cid_num").equals(agentNumber)) {
                    matchFound = true;
                }
            }
```

```
            rs2.close();
            if(!matchFound) {
                activeAgents.add(new Call(agentNumber, null, queueName.substring
                    (0, 1).toUpperCase() + queueName.substring(1), null));
            }
        }
        rs1.close();
    }
    catch(SQLException e) {
        System.out.println(e);
    }
    finally {
        try {
            if(sh.getPreparedStatement() != null) {
                sh.getPreparedStatement().close();
            }
        }
        catch(SQLException e) {
            System.out.println(e);
        }
    }
    return activeAgents;
}
...
```

Výpis 16: Metoda getActiveAgents

C.5 Zdrojový kód třídy k navázání spojení s databází

```
public class DatabaseConnection {
    private DataSource dataSource;

    public DatabaseConnection(String databaseName) {
        try {
            Context initContext = new InitialContext();
            Context envContext = (Context) initContext.lookup("java:/comp/env");
            dataSource = (DataSource) envContext.lookup("jdbc/" + databaseName);
        }
        catch(NamingException e) {
            System.out.println(e);
        }
    }
    public DataSource getDataSource() {
        return dataSource;
    }
}
```

Výpis 17: Třída pro navázaní spojení s databázi

C.6 Zdrojový kód třídy pro zpracování SQL dotazů

```
public class SqlHelper {
    private final Connection connection;
    private PreparedStatement ps;

    public SqlHelper(Connection connection) {
        this.connection = connection;
    }

    public ResultSet getResultSet(String query) throws SQLException {
        ResultSet rs = null;

        try {
            ps = connection.prepareStatement(query);
            rs = ps.executeQuery();
        }
        catch(SQLException e) {
            System.out.println(e);
        }
        return rs;
    }

    public PreparedStatement getPreparedStatement() {
        return ps;
    }
}
```

Výpis 18: Třída pro zpracování SQL dotazů

C.7 Zdrojový kód třídy obsahující informace o hovorech

```
public class Call {
    private String agentChanname;
    private String callerChanname;
    private String queueName;
    private String linkedid;

    public Call(String agentChanname, String callerChanname, String queueName,
        String linkedid) {
        this.agentChanname = agentChanname;
        this.callerChanname = callerChanname;
        this.queueName = queueName;
        this.linkedid = linkedid;
    }
    public String getAgentChanname() {
        return agentChanname;
    }

    public String getCallerChanname() {
        return callerChanname;
    }

    public String getQueueName() {
        return queueName;
    }

    public String getLinkedid() {
        return linkedid;
    }
}
```

Výpis 19: Třída pro vytváření instancí s informacemi o hovorech

C.8 Ukázka zdrojového kódu pro výpis aktivních agentů

```
...
if(dashboard.getActiveAgents().size() > 0) {
    for (int i = 0; i < dashboard.getActiveAgents().size(); i++) {
        content += "<tr>";
        content += "<td><h4 align=\"center\">" + dashboard.getActiveAgents().
            get(i).getAgentChanname() + "</h4></td>";

        boolean matchFound = false;
        for (int j = 0; j < dashboard.getAgentsOnCall().size(); j++) {
            if(dashboard.getActiveAgents().get(i).getAgentChanname().equals(
                dashboard.getAgentsOnCall().get(j).getAgentChanname())) {
                matchFound = true;
                content += "<td><img src=\"images/red_dot_1.png\" style=\"height
                    :30px\"></td>";
            }
        }

        if(!matchFound) {
            content += "<td><img src=\"images/green_dot_1.png\" style=\"height
                :30px\"></td>";
        }
        content += "<td><h4 align=\"center\">" + dashboard.getActiveAgents().
            get(i).getQueueName() + "</h4></td>";
        content += "</tr>";
    }
}
print.out(content);
```

Výpis 20: Výpis seznamu aktivních agentů

C.9 Konfigurace konektivity webové aplikace s MySQL serverem

```
<Context path="/asterisk">
<Resource
   name="jdbc/asterisk"
   auth="Container"
   driverClassName="com.mysql.jdbc.Driver"
   logAbandoned="true"
   maxActive="100"
   maxIdle="30"
   maxWait="10000"
   type="javax.sql.DataSource"
   url="jdbc:mysql://192.168.1.34:3306/asterisk?serverTimezone=Europe/Prague"
   username="asterisk"
   password="asterisk"/>
</Context>
```

Výpis 21: Konfigurace v souboru context.xml

D Obsah SD karty

- Adresář *Asterisk Configuration* s konfigurací systému Asterisk a nastavení ODBC

  `/etc/asterisk/cel.conf`
  `/etc/asterisk/cel_odbc.conf`
  `/etc/asterisk/extensions.conf`
  `/etc/asterisk/queues.conf`
  `/etc/asterisk/sip.conf`
  `/etc/asterisk/voicemail.conf`
  `/etc/odbc.ini`
  `/etc/odbcinst.ini`
  `/var/lib/asterisk/sounds/billing_message.ul`
  `/var/lib/asterisk/sounds/ivr_promt_user.ul`
  `/var/lib/asterisk/sounds/sales_message.ul`
  `/var/lib/asterisk/sounds/tech_message.ul`

- Adresář *Asterisk Monitor* obsahující kompletní NetBeans projekt webové aplikace
- Soubor *README.txt* s instrukcemi k webové aplikaci